IC Compiler
Technology File and Routing Rules Reference Manual

Version C-2009.06, June 2009

SYNOPSYS®

Copyright Notice and Proprietary Information

Right to Copy Documentation

The license agreement with Synopsys permits licensee to make copies of the documentation for its internal use only. Each copy shall include all copyrights, trademarks, service marks, and proprietary rights notices, if any. Licensee must assign sequential numbers to all copies. These copies shall contain the following legend on the cover page:

> "This document is duplicated with the permission of Synopsys, Inc., for the exclusive use of _____ and its employees. This is copy number _____."

Destination Control Statement

All technical data contained in this publication is subject to the export control laws of the United States of America. Disclosure to nationals of other countries contrary to United States law is prohibited. It is the reader's responsibility to determine the applicable regulations and to comply with them.

Disclaimer

SYNOPSYS, INC., AND ITS LICENSORS MAKE NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

Registered Trademarks (®)

Synopsys, AMPS, Astro, Behavior Extracting Synthesis Technology, Cadabra, CATS, Certify, CHIPit, Design Compiler, DesignWare, Formality, HDL Analyst, HSIM, HSPICE, Identify, iN-Phase, Leda, MAST, ModelTools, NanoSim, OpenVera, PathMill, Physical Compiler, PrimeTime, SCOPE, Simply Better Results, SiVL, SNUG, SolvNet, Syndicated, Synplicity, Synplify, Synplify Pro, Synthesis Constraints Optimization Environment, TetraMAX, the Synplicity logo, UMRBus, VCS, Vera, and YIELDirector are registered trademarks of Synopsys, Inc.

Trademarks (™)

AFGen, Apollo, Astro-Rail, Astro-Xtalk, Aurora, AvanWaves, BEST, Columbia, Columbia-CE, Confirma, Cosmos, CosmosLE, CosmosScope, CRITIC, CustomSim, DC Expert, DC Professional, DC Ultra, Design Analyzer, Design Vision, DesignerHDL, DesignPower, DFTMAX, Direct Silicon Access, Discovery, Eclypse, Encore, EPIC, Galaxy, Galaxy Custom Designer, HANEX, HAPS, HapsTrak, HDL Compiler, Hercules, Hierarchical Optimization Technology, High-performance ASIC Prototyping System, HSIM^{plus}, i-Virtual Stepper, IICE, in-Sync, iN-Tandem, Jupiter, Jupiter-DP, JupiterXT, JupiterXT-ASIC, Liberty, Libra-Passport, Library Compiler, Magellan, Mars, Mars-Rail, Mars-Xtalk, Milkyway, ModelSource, Module Compiler, MultiPoint, Physical Analyst, Planet, Planet-PL, Polaris, Power Compiler, Raphael, Saturn, Scirocco, Scirocco-i, Star-RCXT, Star-SimXT, System Compiler, System Designer, Taurus, TotalRecall, TSUPREM-4, VCS Express, VCSi, VHDL Compiler, VirSim, and VMC are trademarks of Synopsys, Inc.

Service Marks (^{SM})

MAP-in, SVP Café, and TAP-in are service marks of Synopsys, Inc.

SystemC is a trademark of the Open SystemC Initiative and is used under license.
ARM and AMBA are registered trademarks of ARM Limited.
Saber is a registered trademark of SabreMark Limited Partnership and is used under license.
All other product or company names may be trademarks of their respective owners.

Contents

Preface

This preface includes the following sections:

What's New in This Release

Information about new features, enhancements, and changes, along with known problems and limitations and resolved Synopsys Technical Action Requests (STARs), is available in the *IC Compiler Release Notes* in SolvNet.

To see the *IC Compiler Release Notes,*

1. Go to the Download Center on SolvNet located at the following address:

   https://solvnet.synopsys.com/DownloadCenter

   If prompted, enter your user name and password. If you do not have a Synopsys user name and password, follow the instructions to register with SolvNet.
2. Select IC Compiler, and then select a release in the list that appears.

About This Manual

The *IC Compiler Technology File and Routing Design Rules Reference Manual* provides details about the technology file and how to implement routing design rules. IC Compiler uses this information to perform placement and routing in IC Compiler.

Audience

Users of this manual must be familiar with IC implementation concepts, specifically, technology specifications, design rule constraints, and standard-cell-based design.

Related Publications

For additional information about IC Compiler, see Documentation on the Web, which is available through SolvNet at the following address:

https://solvnet.synopsys.com/DocsOnWeb

Conventions

The following conventions are used in Synopsys documentation.

Table 1

Convention	Description
`Courier`	Indicates command syntax.
`Courier italic`	Indicates a user-defined value in Synopsys syntax, such as *`object_name`*. (A user-defined value that is not Synopsys syntax, such as a user-defined value in a Verilog or VHDL statement, is indicated by regular text font italic.)
`Courier bold`	Indicates user input—text you type verbatim—in Synopsys syntax and examples. (User input that is not Synopsys syntax, such as a user name or password you enter in a GUI, is indicated by regular text font bold.)
[ ]	Denotes optional parameters, such as *`pin1`* [*`pin2 ... pinN`*]
|	Indicates a choice among alternatives, such as `low | medium | high` (This example indicates that you can enter one of three possible values for an option: low, medium, or high.)
_	Connects terms that are read as a single term by the system, such as `set_annotated_delay`
Control-c	Indicates a keyboard combination, such as holding down the Control key and pressing c.
\	Indicates a continuation of a command line.
/	Indicates levels of directory structure.
Edit > Copy	Indicates a path to a menu command, such as opening the Edit menu and choosing Copy.

Customer Support

Customer support is available through SolvNet online customer support and through contacting the Synopsys Technical Support Center.

Accessing SolvNet

SolvNet includes an electronic knowledge base of technical articles and answers to frequently asked questions about Synopsys tools. SolvNet also gives you access to a wide range of Synopsys online services including software downloads, documentation on the Web, and “Enter a Call to the Support Center.”

To access SolvNet, go to the SolvNet Web page at the following address:

https://solvnet.synopsys.com

If prompted, enter your user name and password. If you do not have a Synopsys user name and password, follow the instructions to register with SolvNet.

If you need help using SolvNet, click HELP in the top-right menu bar or in the footer.

Contacting the Synopsys Technical Support Center

If you have problems, questions, or suggestions, you can contact the Synopsys Technical Support Center in the following ways:

- Open a call to your local support center from the Web by going to http://solvnet.synopsys.com (Synopsys user name and password required), and then clicking “Enter a Call to the Support Center.”
- Send an e-mail message to your local support center.
  - E-mail support_center@synopsys.com from within North America.
  - Find other local support center e-mail addresses at http://www.synopsys.com/Support/GlobalSupportCenters/Pages
- Telephone your local support center.
  - Call (800) 245-8005 from within the continental United States.
  - Call (650) 584-4200 from Canada.
  - Find other local support center telephone numbers at http://www.synopsys.com/Support/GlobalSupportCenters/Pages

1

Technology File Syntax

A technology file provides technology-specific information, such as the names and physical and electrical characteristics of each metal layer and the routing design rules. This chapter describes the syntax of the technology file. After you create a technology file, you associate it with a Milkyway design library. IC Compiler uses the Milkyway design library to access the technology information.

This chapter provides instructions for creating and loading a technology file and information about each of the technology file sections. A technology file contains process-specific information such as layer thicknesses and the sheet resistance of the various layers.

The chapter is organized into the following sections:

- Tile Section
- Layer Section
- ContactCode Section
- DesignRule Section
- FringeCap Section
- CapModel and CapTable Sections
- ResModel Section
- PRRule Section
- DensityRule Section
- SlotRule Section

Creating a Technology File

You can create a technology file by using the syntax descriptions in this chapter or by editing an existing technology file.

To generate an editable version of the technology file associated with a Milkyway design library, use the `write_mw__lib_files -technology` command (or choose File > Export > Write Library File in the GUI). You specify the name of the generated technology file by using the `-output` option.

```
icc_shell> write_mw_lib_files -technology -output techfile.tf
```

Basic Syntax Rules

These are the basic technology file syntax rules:

- The file extension used for an editable technology file is .tf.
- A technology file consists of several sections, each of which follows the following syntax:

```
section_name{
      attribute_name = attribute_value...
}
```

- Keywords, such as section and attribute names, are case-sensitive and must be typed with the exact spelling and capitalization given. They are reserved words and cannot be used outside the specified context.
- Parentheses ( ) should be typed as they appear in the syntax.
- You can add comments to the technology file by placing a slash and an asterisk (/*) at the beginning of a comment and an asterisk and a slash (*/) at the end. All text between /* and */ is ignored.

Technology File Contents

Example 1-1 shows the contents of the technology file. The following sections describe each of the technology file sections.

Example 1-1 Technology File Contents

```
/* Specifies units and unit values */
Technology {
  units
  operating_conditions
  routing_rule_modes
```

```
}

/* Defines the six basic colors used to create display colors */
[PrimaryColor {
  primarycolor_attributes
}]

/* Defines the custom display colors used to display designs in the
library */
Color color_value {
  color_attributes
}...

/* Defines the stipple patterns used to display designs in the library */
Stipple "name" {
  stipple_attributes
}...

/* Defines the line styles used to display designs in the library */
LineStyle "name" {
  linestyle_attributes
}...

/* Defines the unit tiles */
Tile "name" {
  tile_attributes
}...

/* Defines the layer-specific characteristics, including display
characteristics
      and layer-specific routing design rules */
Layer "name" {
  display_attributes
  layout_attributes
  parasitic_attributes
  physical_attributes
}...


/* Defines the via masters used in designs in the library */
ContactCode "name" {
  contactcode_attributes
}...

/* Defines the interlayer routing design rules that apply to designs in
the library */
DesignRule {
  layer_attributes
  rule_attributes
}...


/* Defines capacitance information for interconnect layers */
```

```
FringeCap value {
  fringecap_attributes
}...

/* Defines timing information */
CapModel {
  capmodel_attributes
}

/* Defines timing information */
CapTable {
  captable_attributes
}

/* Defines the resistance and temperature coefficient of a layer as a
function of wire
width */
ResModel {
  resmodel_attributes
}

/* Defines cell row spacing rules */
PRRule {
  prrrule_attributes
}...

/* Defines density rules */
DensityRule {
  densityrule_attributes
}...


/* Defines slot rules */
SlotRule {
  slotrule_attributes
}...
```

Technology Section

Use the `Technology` section of a Milkyway technology file to specify global attributes, such as units and routing rule modes. Example 1-2 shows a sample `Technology` section.

Example 1-2 Technology Section

```
Technology {
     /* define units */
     dielectric = 0.000000e+00
     unitLengthName = "micron"
     lengthPrecision = 1000
     gridResolution = 50
     unitTimeName = "ns"
     timePrecision = 100
     unitCapacitanceName = "pf"
     capacitancePrecision = 10000
     unitResistanceName = "kohm"
     resistancePrecision = 10
     unitInductanceName = "nh"
     inductancePrecision = 1000
     unitPowerName = "mw"
     powerPrecision = 10000
     unitVoltageName = "V"
     voltagePrecision = 1000
     unitCurrentName = "mA"
     currentPrecision = 1000000


     /* define routing rule modes */
     minLengthMode = 0
     minAreaMode = 0
     fatTblMinEnclosedArea = 0
     minEdgeMode = 0
     cornerSpacingMode = 0
     fatTblSpacingMode = 0
     parallelLengthMode = 0
     fatWireExtensionMode = 0
     stubMode = 0
     maxStackLevelMode = 1
}
```

Unit Precision and Range

You specify the unit size and precision of each unit (length, resistance, capacitance, and so on) in the `Technology` section. Unit values are stored as 32-bit integers, so the specified unit size and precision determine the maximum dynamic range of the unit in the design.

The range of integers that can be stored is from -2^{31} to $2^{31} - 1$ (–2,147,483,648 to +2,147,483,647). For example, if you set the unit length specification to micron and the length precision specification to 1,000, the maximum dynamic range is from –2,147,483.648 to +2,147,483.647 microns.

If you want to measure capacitance down to 0.0001 picofarad (pF), you need to set unit capacitance to pF and capacitance precision to 10,000. In that case, the maximum capacitance that can be measured is 214,748 pF.

Defining Units

Use the following attributes to define the units in the `Technology` section:

`dielectric`

The `dielectric` attribute specifies the relative permittivity of SiO_2 that is to be used to calculate sidewall capacitance with a dielectric statement in the technology file.

The tool determines the dielectric unit by dividing `unitCapacitanceName` by `unitLengthName`.

`unitLengthName`

The `unitLengthName` attribute defines the linear distance unit. The valid values are

- `micron`
- `mil`

`lengthPrecision`

The `lengthPrecision` attribute sets the number of database units per user unit, thereby defining the precision of distance measurements stored in the database.

For example, if you set `unitLengthName` to micron and `lengthPrecision` to 1,000, the database unit will be 0.001 micron. Thus, all distance measurements are rounded to the nearest 0.001 micron.

`gridResolution`

The `gridResolution` attribute sets the manufacturing grid resolution in database units. For example, if you set `unitLengthName` to micron and `lengthPrecision` to 1,000, the database unit is 0.001 micron. If the minimum grid resolution of the manufacturing process is 0.1 micron, `gridResolution` should be set to 100.

When you place an object with IC Compiler, the object's point of origin falls on the grid.

`unitTimeName`

The `unitTimeName` attribute defines the time unit.

Table 1-1 shows the valid units for `unitTimeName`.

Table 1-1 Valid Units for unitTimeName

Unit	Value
`fs`	1 x 10^{-15} second
`ps`	1 x 10^{-12} second
`ns`	1 x 10^{-9} second
`us`	1 x 10^{-6} second
`ms`	1 x 10^{-3} second
`s`	second

`timePrecision`

The `timePrecision` attribute defines the precision of time measurements stored in the database.

For example, if you set `unitTimeName` to `ns` and `timePrecision` to 100, the database unit is 0.01 ns. Thus, all time measurements are rounded to the nearest 0.01 ns.

`unitCapacitanceName`

The `unitCapacitanceName` attribute defines the capacitance unit.

Table 1-2 shows the valid units for `unitCapacitanceName`.

Table 1-2 Valid Units for unitCapacitanceName

Unit	Value
`ff`	1 x 10^{-15} farad
`pf`	1 x 10^{-12} farad
`nf`	1 x 10^{-9} farad
`uf`	1 x 10^{-6} farad
`mf`	1 x 10^{-3} farad
`f`	farad

`capacitancePrecision`

The `capacitancePrecision` attribute defines the precision of capacitance measurements stored in the database.

For example, if you set `unitCapacitanceName` to `pf` and `capacitancePrecision` to 10,000, the database unit is 0.0001 picofarad or 0.1 femtofarad. Thus, all capacitance measurements are rounded to the nearest 0.1 femtofarad.

`unitResistanceName`

The `unitResistanceName` attribute defines the resistance unit.

Table 1-3 shows the valid units for `unitResistanceName`.

Table 1-3 Valid Units for unitResistanceName

Unit	Value
`mohm`	1 x 10^{-3} ohm
`ohm`	ohm
`kohm`	1 x 10^{3} ohm
`Mohm`	1 x 10^{6} ohm

`resistancePrecision`

The `resistancePrecision` attribute defines the precision of resistance measurements stored in the database.

For example, if you set `unitResistanceName` to `ohm` and `resistancePrecision` to 1,000, the database unit is 0.001 ohm. Thus, all resistance measurements are rounded to the nearest 0.001 ohm.

`unitInductanceName`

The `unitInductanceName` attribute defines the inductance unit.

Table 1-4 shows the valid units for `unitInductanceName`.

Table 1-4 Valid Units for unitInductanceName

Unit	Value
`fH`	1 x 10^{-15} henry
`pH`	1 x 10^{-12} henry

Table 1-4 Valid Units for unitInductanceName (Continued)

Unit	Value
nH	1 x 10^{-9} henry
uH	1 x 10^{-6} henry
mH	1 x 10^{-3} henry
H	henry

inductancePrecision

The inductancePrecision attribute defines the precision of inductance measurements stored in the database.

For example, if you set unitInductanceName to nH and inductancePrecision to 1,000, the database unit will be 0.001 nanohenry (nH). Thus, all inductance measurements are rounded to the nearest 0.001 nH.

unitPowerName

The unitPowerName attribute defines the power unit.

Table 1-5 shows the valid units for unitPowerName.

Table 1-5 Valid Units for unitPowerName

Unit	Value
fw	1 x 10^{-15} watt
pw	1 x 10^{-12} watt
nw	1 x 10^{-9} watt
uw	1 x 10^{-6} watt
mw	1 x 10^{-3} watt
w	watt

powerPrecision

The powerPrecision attribute defines the precision of power measurements stored in the database.

For example, if you set `unitPowerName` to `mw` and `powerPrecision` to 1,000, the database unit is 0.001 milliwatt (mw). Thus, all power measurements are rounded to the nearest 0.001 mw.

`unitVoltageName`

The `unitVoltageName` attribute defines the voltage unit.

Table 1-6 shows the valid units for `unitVoltageName`.

Table 1-6 Valid Units for unitVoltageName

Unit	Value
`fV`	1 x 10^{-15} volt
`pV`	1 x 10^{-12} volt
`nV`	1 x 10^{-9} volt
`uV`	1 x 10^{-6} volt
`mV`	1 x 10^{-3} volt
`V`	volt

`voltagePrecision`

The `voltagePrecision` attribute defines the precision of voltage measurements stored in the database.

For example, if you set `unitVoltageName` to `mV` and `voltagePrecision` to 1,000, the database unit is 0.001 millivolt (mV). Thus, all power measurements are rounded to the nearest 0.001 mV.

`unitCurrentName`

The `unitCurrentName` attribute defines the current unit.

Table 1-7 shows the valid units for `unitCurrentName`.

Table 1-7 Valid Units for unitCurrentName

Unit	Value
`fA`	1 x 10^{-15} ampere
`pA`	1 x 10^{-12} ampere

Table 1-7 Valid Units for unitCurrentName (Continued)

Unit	Value
nA	1 x 10^{-9} ampere
uA	1 x 10^{-6} ampere
mA	1 x 10^{-3} ampere
A	ampere

currentPrecision

The currentPrecision attribute defines the precision of current measurements stored in the database.

For example, if you set unitCurrentName to milliampere (mA) and currentPrecision to 1,000, the database unit is 0.001 mA. Thus, all power measurements are rounded to the nearest 0.001 mA.

Defining Routing Rule Modes

In the Technology section, use the following attributes to define the routing rule modes.

minLengthMode

Determines whether via cuts are considered in applying the minimum length rule. Table 1-8 shows the valid values for the minLengthMode attribute.

Table 1-8 Valid Values for minLengthMode

Mode	Description
0 (default)	IC Compiler uses the total wire length to check the minimum length rule.
1	IC Compiler uses wire segments that include via cuts to check the minimum length.

For more information, see “Minimum Length Rule” on page 2-3.

`minAreaMode`

Determines whether the shapes of areas are considered in applying the minimum area rule. Table 1-9 shows the valid values for the `minAreaMode` attribute.

Table 1-9 Valid Values for minAreaMode

Mode	Description
0 (default)	Ignores the `specialMinArea` value and honors the `minArea` rule.
1	Honors the `specialMinArea` value if the polygon is not a rectangle.

For more information, see "Two-Stage Special Minimum Area Rule" on page 2-5.

`fatTblMinEnclosedAreaMode`

Determines how to apply the minimum enclosed area rule. Table 1-10 shows the valid values for the `fatTblMinEnclosedAreaMode` attribute.

Table 1-10 Valid Values for fatTblMinEnclosedAreaMode

Mode	Description
0 (default)	The fat wire minimum enclosed area mode is triggered when any of the surrounding metal satisfies the width requirement.
1	The fat wire minimum enclosed area mode is triggered only when all of the surrounding metal satisfies the width requirement.

For more information, see "Minimum Enclosed Area Rule" on page 2-6.

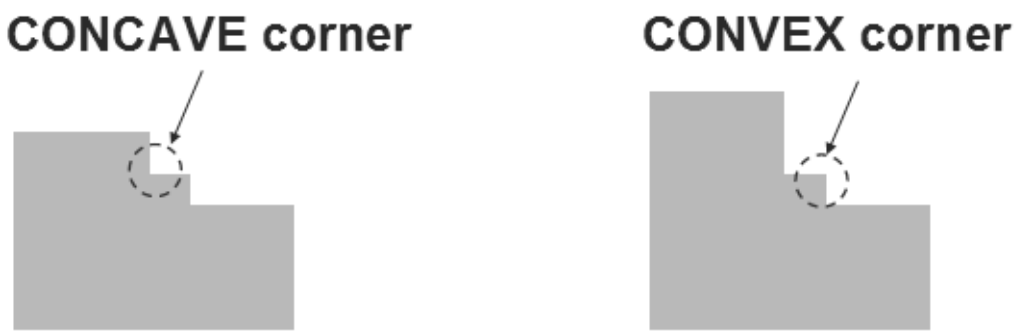
`minEdgeMode`

Controls the scope of the check for the minimum edge rules. Table 1-11 shows the valid values for the `minEdgeMode` attribute.

Table 1-11 Valid Values for minEdgeMode

Mode	Description
0 (default)	Needs a concave corner to trigger a violation. A concave corner is formed by two adjacent edges when both are less than minimum length. Figure 1-1 shows examples of convex and concave corners.
1	The minimum edge length is violated if the total number of consecutive minimum edges is greater than the value specified by `maxNumMinEdge` or if the total edge length is greater than `maxTotalMinEdgeLength`.

Figure 1-1 Concave and Convex Corners



For more information, see "Minimum Edge Rules" on page 2-8.

`cornerSpacingMode`

Specifies whether diagonal or rectilinear distance measurement is used for via spacing. If this attribute is set to 0, diagonal measurement is used; the actual spacing, measured diagonally, must satisfy the via minimum spacing requirement. If it is set to 1, rectilinear

measurement is used; at least one of the X and Y directions must satisfy the via minimum spacing requirement. Table 1-12 shows the valid values for the `cornerSpacingMode` attribute.

Table 1-12 Valid Values for cornerSpacingMode

Mode	Description
0 (default)	The corner-to-corner via spacing is measured by using the diagonal distance.
1	The corner-to-corner via spacing is measured by using the Manhattan, rectilinear distance.

For more information, see "Via Corner Spacing Rule" on page 2-15.

`fatTblSpacingMode`

Determines the effect of the parallel length (`fatTblParallelLength` attribute) on the indexing of the `fatTblSpacing` attribute. Table 1-13 shows the valid values for the `fatTblSpacingMode` attribute.

Table 1-13 Valid Values for fatTblSpacingMode

Mode	Description
0 (default)	Downgrade the `fatTblSpacing` index according to the value of the `fatTblParallelLength` attribute.
1	Downgrade the `fatTblSpacing` index by a maximum of one.

For more information, see "Fat Metal Spacing Rule Table" on page 2-24.

`parallelLengthMode`

The `parallelLengthMode` attribute controls the merging of wire segments based on the parallel length rule. Table 1-14 shows the valid values for the `parallelLengthMode` attribute.

Table 1-14 Valid Values for parallelLengthMode

Mode	Description
0 (default)	Merge wire segments only with fat neighboring shapes.

Table 1-14 Valid Values for parallelLengthMode

Mode	Description
1	Merge wire segments only with all neighboring shapes.

For more information, see "Fat Metal Parallel Length" on page 2-28.

`fatWireExtensionMode`

The `fatWireExtensionMode` attribute controls the application of the fat metal extension spacing rule. Table 1-15 shows the valid values for the `fatWireExtensionMode` attribute.

Table 1-15 Valid Values for fatWireExtensionMode

Mode	Description
0 (default)	The fat spacing check is performed only on extension wires connected to the fat wire and within the extension range from the fat wire edges and corners. The spacing is checked between any two wires. Projection length is not checked.
1	The fat spacing check is performed on both connected and unconnected wires within the extension range from the fat wire edges, not including fat wire corners. Spacing between both overlapped and non-overlapped wire pairs is checked. Projection lengths are also checked.
2	The fat spacing check is performed on both connected and unconnected wires within the extension range from the fat wire edges, not including fat wire corners. Only the spacing between non-overlapped wire pairs is checked. Projection lengths are not checked.
3	The fat spacing check is performed on both connected and unconnected wires within the extension range from the fat wire edges, not including fat wire corners. Only the spacing between overlapped wire pairs is checked. Projection lengths of these wires are also checked.
4	The fat spacing check is performed only on connected wires within the extension range from the fat wire edges, not including fat wire corners. The spacing from these wires to all other wires is checked. Projection lengths are not checked.

For more information, see "Fat Metal Extension Spacing Rule" on page 2-29.

`stubMode`

Specifies when the stub spacing rule is applied, rather than the minimum spacing rule. Table 1-16 shows the valid values for the `stubMode` attribute.

Table 1-16 Valid Values for stubMode

Mode	Description
0 (default)	Stub spacing is used when the distance that two objects run parallel to each other is less than or equal to `stubThreshold`.
1	Stub spacing is used when an end-of-line metal segment has an edge width that is less than or equal to `stubThreshold`.
2	Stub spacing is used when a metal segment with a width less than or equal to `stubThreshold` has neighboring metal along two adjacent edges or any one edge, which is less than `stubThreshold` from the corner of two adjacent edges.
3	Stub spacing is used when a metal with a width less than or equal to `stubThreshold` has neighboring metals along three adjacent edges and has neighboring metal along two adjacent edges within `stubThreshold`.
4	Stub spacing is used when a metal segment has a width that is less than `stubThreshold` and there is no connecting metal within `minWidth`, and if it has neighboring metal along two adjacent edges or any one edge that is less than `stubThreshold` from the corner of two adjacent edges.

For more information, see "End-of-Line Spacing Rules" on page 2-38.

`maxStackLevelMode`

Controls the scope of the via array maximum stack level rule. Table 1-17 shows the valid values for the `maxStackLevelMode` attribute.

Table 1-17 Valid Values for maxStackLevelMode

Mode	Description
0 (default)	Ignore the maximum stack level rule check when all stacked vias are via arrays.
1	Check the maximum stack level rule for stacked via arrays.
2	Ignore the maximum stack level rule check when at least one stacked via is a via array.
3	Ignore the maximum stack level rule check when all stacked vias are aligned via arrays (overlapped with at least two cuts).

For more information, see "Via Maximum Stack Level Rule" on page 2-75.

PrimaryColor Section

The `PrimaryColor` section of the technology file defines the six basic colors that IC Compiler uses to create display colors.

To understand how the system combines primary colors to create display colors, you need to understand how a computer monitor creates colors on the screen.

Colors on a Computer Monitor

In any system, the term *primary colors* refers to the colors used to define all other colors. A computer monitor has three primary colors; IC Compiler has six.

All colors displayed on a computer monitor are created by combining red, green, and blue light. Combining colors of light produces results that are very different from the results that come from combining the same colors of paint. Before you decide to change your colors, note the following results that come from combining colors of light:

- green + red = yellow
- green + blue = cyan

- blue + red = magenta

The intensity of any primary color on a computer monitor (red, green, or blue) can be from 0 to 255, with 255 producing the brightest intensity of the color.

If you are working with computer graphics for the first time, you might find that combining colors can produce unexpected results.

Colors in IC Compiler

The `PrimaryColor` section of a technology file defines the six primary colors for IC Compiler: two intensities of red (`lightRed` and `mediumRed`), two intensities of green (`lightGreen` and `mediumGreen`), and two intensities of blue (`lightBlue` and `mediumBlue`). The intensity value is a number between 0 and 255.

IC Compiler uses combinations of these six primary colors to create the display colors you see when you display a cell in the layout window. If you do not include the `PrimaryColor` section in the technology file, IC Compiler uses the default values shown in Example 1-3.

Example 1-3 shows a sample `PrimaryColor` section.

Example 1-3 Defining the Primary Colors

```
PrimaryColor {
      lightRed = 90
      mediumRed = 180
      lightGreen = 80
      mediumGreen = 175
      lightBlue = 100
      mediumBlue = 190
}
```

Color Section

The primary colors defined in the `PrimaryColor` section (or the default primary colors if the technology file does not contain a `PrimaryColor` section) determine the default display colors. There are 64 display colors, which are stored as 6-bit binary numbers. Each bit represents one of the primary colors, as follows:

Medium red	Light red	Medium green	Light green	Medium blue	Light blue

For example, color number 51, stored as the binary number 110011, consists of the primary colors represented by the bits that are set, including the following:

- Medium red
- Light red
- Medium blue
- Light blue

As described in Example 1-3, the two intensities for red are

```
lightRed 90
mediumRed 180
```

If you add the two intensities together, the result is 270. Because a monitor cannot produce an intensity greater than 255, a display color that combines light red and medium red is red in an intensity of only 255.

In the case of color number 51, the combination of medium red and light red exceeds the maximum intensity of red the monitor can produce. Likewise, the combination of medium blue and light blue exceeds the maximum intensity of blue the monitor can produce. Therefore, creating color number 51 means combining red at an intensity of 255 and blue at an intensity of 255.

You can override the default display color for any color number by defining a custom display color in a `Color` section of the technology file.

The display colors define the colors that IC Compiler uses to display designs in the library. The `Layer` section uses the display colors to specify how layers are displayed.

Note:

When two display colors overlap, the color produced is the result of an OR operation between the two color numbers. For example, if color number 51 (110011) overlaps color number 56 (111000), the result is color number 59 (111011).

You define a custom display color by defining the following attributes in the `Color` section:

`name`

Specifies the name of the color. The name is a user-defined string of up to 31 characters.

`rgbDefined`

Indicates whether the color is red, green, blue (RGB)-defined. Valid values are 1 or 0.

`redIntensity`

Specifies the color's red intensity. Valid values are integers from 0 to 255.

`greenIntensity`

Specifies the color's green intensity. Valid values are integers from 0 to 255.

`blueIntensity`

Specifies the color's blue intensity. Valid values are integers from 0 to 255.

Example 1-4 shows a `Color` section that defines color 62 ("owhite"):

Example 1-4 Defining a Color

```
Color 62 {
      name = "owhite"
      rgbDefined = 1
      redIntensity = 255
      greenIntensity = 255
      blueIntensity = 230
}
```

Stipple Section

The `Stipple` section of the technology file defines the stipple patterns IC Compiler uses to display designs in the library. The `Layer` section uses the stipple patterns to specify how layers are displayed.

You define a stipple pattern by defining the following attributes in the `Stipple` section:

`name`

Specifies a name that identifies the stipple pattern. The name is a user-defined string of up to 15 characters.

`width`

Specifies the horizontal dimension of the stipple pattern, in pixels.

`height`

Specifies the vertical dimension of the stipple pattern, in pixels.

`pattern`

Defines the stipple pattern, with 1s representing pixels drawn with the color assigned to the layer and 0s representing pixels with no color (transparent). The pattern representation is enclosed in parentheses.

You need to create a `Stipple` section for each required pattern.

Example 1-5 shows the definition for a stipple pattern named "blank." Example 1-6 shows the definition for a stipple pattern named "solid."

Example 1-5 Specifying a Blank Stipple Pattern

```
Stipple "blank" {
```

```
      width = 2
      height = 2
      pattern = (0, 0, 0, 0)
}
```

Example 1-6 Specifying a Solid Stipple Pattern

```
Stipple "solid" {
      width = 2
      height = 2
      pattern = (1, 1, 1, 1)
}
```

LineStyle Section

The `LineStyle` section of the technology file defines the line styles IC Compiler uses to display designs in the library. The `Layer` section uses the line styles to determine how layers are displayed.

You define a line style by defining the following attributes in the `LineStyle` section:

`name`

Specifies a name that identifies the line style. The name is a user-defined string of up to 15 characters.

`width`

Specifies the horizontal dimension of the line style pattern in pixels.

`height`

Specifies the vertical dimension of the line style pattern in pixels.

`pattern`

Defines the line style pattern, with 1s representing pixels drawn with the color assigned to the layer and 0s representing pixels with no color (transparent). The pattern representation is enclosed in parentheses.

Note:

Because line styles are predefined, they do not actually need to appear in the technology file.

Example 1-7 shows the definition for a line style named "boundary."

Example 1-7 Specifying a Line Style

```
LineStyle "boundary" {
      width = 10
      height = 1
      pattern = (1, 0, 0, 1, 1, 1, 1, 1, 0, 0)
}
```

Tile Section

A `Tile` section defines the unit tiles.

You define a unit tile by defining the following attributes in the `Tile` section:

`name`

Specifies the name of the unit tile. The name is a user-defined string of up to 31 characters.

`width`

Specifies the width of a tile.

`height`

Specifies the height of a tile.

Example 1-8 shows a typical unit tile definition.

Example 1-8 Tile Section of a Technology File

```
Tile "unit" {
      width = 16
      height = 168
}
```

Layer Section

A technology file defines each layer by specifying attributes for that layer. The layer can be a metal layer or a via layer. The attributes fall into several groupings, including

- Layout attributes

  Layout attributes associate a physical layer in the layout with the display layer.

- Display attributes

  Display attributes specify how objects on the layer are displayed.

- Parasitic attributes

  Parasitic attributes define the layer parasitics for a metal layer.

  Note:
  You define the parasitic attributes for a via layer in a `ContactCode` section.

- Physical attributes

  Physical attributes define physical characteristics of the layer and need to be specified if you are using timing-driven layout.

- Design rule attributes

  Design rule attributes define the layer-specific design rules associated with objects on the layer.

Example 1-9 shows all the attributes for a `Layer` section.

Example 1-9 Layer Section

```
Layer "MET1" {
      /* layout attributes */
     layerNumber = 8
     isDefaultLayer = 0
     maskName = "metal1"
     pitch = 2.2

      /* display attributes */
     color = "blue"
     lineStyle = "solid"
     pattern = "dot"
     blink = 0
     visible = 1
     selectable = 1
     panelNumber = 0

      /* parasitic attributes */
     unitMinResistance = 0
     unitNomResistance = 0
     unitMaxResistance = 0
     unitMinCapacitance = 0
     unitNomCapacitance = 0
     unitMaxCapacitance = 0
     unitMinInductance = 0
     unitNomInductance = 0
     unitMaxInductance = 0
     unitMinSideWallCap = 0
     unitNomSideWallCap = 0
     unitMaxSideWallCap = 0
     unitMinChannelCap = 0
     unitNomChannelCap = 0
     unitMaxChannelCap = 0
     unitMinChannelSideCap = 0
     unitNomChannelSideCap = 0
     unitMaxChannelSideCap = 0
     maxCurrDensity = 0
     maxIntraCapDistRatio = 0
     maxSegLenForRC = 0


      /* physical attributes */
     unitMinHeightFromSub = 0
     unitNomHeightFromSub = 0
     unitMaxHeightFromSub = 0
     unitMinThickness = 0
```

```
unitNomThickness = 0
unitMaxThickness = 0

 /* design rule attributes */
maxWidth = 1.0
minWidth = 1.0
defaultWidth = 1.0
minLength = 1.4
maxLength = 1.4
minArea = 1.0
specialMinArea = 1.0
minAreaEdgeThreshold = 1.0
minEnclosedArea = 1.0
minEnclosedWidth = 1.0
minEdgeLength = 0
maxNumMinEdge = 0
maxTotalMinEdgeLength = 0
minEdgeLength2 = 0
minEdgeLength3 = 0
minSpacing = 1.0
cornerMinSpacing = 1.0
sameNetMinSpacing = 0
checkManhattanSpacing = 0
cutTblSize = 0
cutNameTbl = (name1, name2, name3)
cutWidthTbl = (0, 0, 0)
cutHeightTbl = (0, 0, 0)
cutDataTypesTble = (0, 0, 0)
maxNumAdjacentCut = 0
enclosedCutNumNeighbor = 1.0
enclosedCutNeighborRange = 1.0
enclosedCutMinSpacing = 1.0
enclosedCutToNeighborMinSpacing = 1.0
onGrid = 0
endOfLine1NeighborEndToEndThreshold = 0
endOfLine1NeighborEndToEndThreshold2 = 0
endOfLine1NeighborEndToEndMinLength = 0
endOfLine1NeighborEndToEndParallelWidth = 0
endOfLine1NeighborEndToEndMinSpacing = 0
endOfLine1NeighborCornerKeepoutWidth = 0
endOfLine2NeighborThreshold = 0
endOfLine2NeighborMinSpacing = 0
endOfLine2NeighborSideMinSpacing = 0
endOfLine2NeighborCornerKeepoutWidth = 0
endOfLine2NeighborSideKeepoutLength = 0
endOfLine3NeighborThreshold = 0
endOfLine3NeighborMinSpacing = 0
endOfLine3NeighborSideMinSpacing = 0
endOfLine3NeighborCornerKeepoutWidth = 0
endOfLine3NeighborSideKeepoutLength = 0
stubThreshold = 0
stubLengthThreshold = 0
stubSpacing = 0
```

```
        stubToStubSpacing = 0
        endOfLineCornerKeepoutWidth = 0
        stubMinLength = 0
        tJunctionStubWireMaxThreshold   = 0
        tJunctionOrthoWireMaxThreshold  = 0
        tJunctionStubKeepoutMinSpacing  = 0
        tJunctionStubKeepoutMinWidth    = 0
        uShapeMinSpacing = 0
        uShapeDepthThreshold = 0
        uShapeMinLength = 0
        sameNetWidthThreshold = 0
        maxStackLevel = 0
        fatWireThreshold = 0
        fatFatMinSpacing = 0
        fatThinMinSpacing = 0
        fatWireExtensionRange = 0
        fatTblDimension = 0
        fatTblThreshold = (0, 0, 0)
        fatTblThreshold2 = (0, 0, 0)
        fatTblParallelLength = (0, 0, 0)
        fatTblExtensionRange = (0, 0, 0)
        fatTblSpacing = (0, 0, 0, 0, 0, 0, 0, 0)
        fatTblMinEnclosedArea = (0, 0, 0)
        fatTblXDimension = 0
        fatTblYDimension = 0
        fatTblXThreshold = (0, 0)
        fatTblYThreshold = (0, 0)
        fatTblXParallelLength = (0, 0)
        fatTblYParallelLength = (0, 0)
        fatTblXMinSpacing = (0, 0, 0, 0)
        fatTblYMinSpacing = (0, 0, 0, 0)
        orthoSpacingExcludeCorner = 0
        fatContactThreshold = (0, 0, 0)
        fatTblFatContactNumber = (0, 0)
        fatTblFatContactMinCuts = (0, 0)
        fat2DTblFatContactNumber = (0, 0, 0, 0)
        fat2DTblFatContactMinCuts = (0, 0, 0, 0)
        fatTblExtensionContactNumber = (0, 0, 0, 0)
        fatTblExtensionMinCuts = (0, 0, 0, 0)
        spanTblDimension      = 0
        spanTblThreshold      = (0, 0, 0, 0)
        spanTblParallelLength = (0, 0, 0, 0)
        spanTblMinSpacing     = (0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)
        protrusionTblDim = 0
        protrusionFatThresholdTbl = (0, 0, 0)
        protrusionLengthLimitTbl = (0, 0, 0)
        protrusionMinWidthTbl = (0, 0, 0)
    }
```

Layout Attributes

The layout attributes associate a physical layer in the layout with the display layer.

The layout attributes are

`layerNumber`

Defines the number that identifies the layer.

Valid values are 1–160 (user defined) and 161–255 (system defined). Layer numbers 188 through 255 are reserved for IC Compiler to create layers for place and route functionality.

`isDefaultLayer`

Specifies the layer used for routing when there are multiple layers with the same `maskName` value.

Valid values are 0 or 1.

Note:
Only one layer with the same mask should be designated as the default.

`maskName`

Defines the physical layer associated with the display layer.

Valid values: In addition to the following predefined mask layers, you can specify any string of up to 31 characters.

- `poly`
- `metal1`... `metal15`
- `polyCont`
- `via1` ... `via15`
- `passivation` (for bonding pad pins)

`pitch`

Defines the predominant separation distance between the centers of objects on the layer.

IC Compiler uses the pitch you specify to generate wire tracks in the unit tile.

Note:
You cannot change the metal2 pitch after data preparation because Milkyway uses the metal2 pitch during blockage, pin, and via (BPV) extraction.

Display Attributes

This section describes the display attributes and their valid values. The display attributes specify how objects on the layer are displayed.

`color`

Defines the name or number of the color used to display the layer.

Valid values are any integer from 0 to 63 or the name of a color that is defined in the `Color` section.

If you specify a number, the color does not have to be defined in the `Color` section. (See "Colors in IC Compiler" on page 1-19.)

`lineStyle`

Sets the defined line style for objects on the layer.

Valid values are the name of a style defined in the `LineStyle` section. (See "LineStyle Section" on page 1-22.)

`pattern`

Defines the stipple pattern used to fill objects on the layer.

Valid values are the name of a stipple pattern defined in the `Stipple` section. (See "Stipple Section" on page 1-21.)

`blink`

Sets the layer to blink or not to blink.

Valid values are 1 (on) or 0 (off).

`visible`

Sets the layer visibility.

Valid values are 1 (on) or 0 (off).

`selectable`

Sets the layer selectability.

Valid values are 1 (on) or 0 (off).

`panelNumber`

Specifies the subpanel for the layer.

Valid values are 0 through 3.

Parasitic Attributes

The parasitic attributes define the metal layer parasitics. In general, IC Compiler gets the parasitic information from the TLUPlus files rather than from the technology file; however, extraction does use the maximum intracapacitance distance ratio and the wire segment length attributes.

This section describes the following parasitic attributes:

- Resistance
- Capacitance
- Inductance
- Sidewall Capacitance
- Routing Channel Capacitance
- Total Sidewall Routing Channel Capacitance
- Maximum Current Density
- Maximum Intracapacitance Distance Ratio
- Wire Segment Length

Note:
For via layers, you specify the parasitic attributes in the `ContactCode` section. (See "ContactCode Section" on page 1-41.)

Resistance

Resistance is defined as the resistance per square (length divided by width) of a poly or metal layer. The resistance attributes are

`unitMinResistance`

A floating-point number representing the minimum resistance.

`unitNomResistance`

A floating-point number representing the nominal resistance.

`unitMaxResistance`

A floating-point number representing the maximum resistance.

Capacitance

Capacitance is defined per square user unit of a poly or metal layer in a cell instance or over a macro. The capacitance attributes are

`unitMinCapacitance`

A floating-point number representing the minimum capacitance.

`unitNomCapacitance`

A floating-point number representing the nominal capacitance.

`unitMaxCapacitance`

A floating-point number representing the maximum capacitance.

Inductance

Inductance is defined per unit length of a poly or metal layer.

Note:
IC Compiler does not use the inductance values.

The inductance attributes are

`unitMinInductance`

A floating-point number representing the minimum inductance.

`unitNomInductance`

A floating-point number representing the nominal inductance.

`unitMaxInductance`

A floating-point number representing the maximum inductance.

Sidewall Capacitance

Sidewall capacitance is defined as the total sidewall capacitance per unit length for both sides of a poly or metal layer in a cell instance or over a macro.

The sidewall capacitance attributes are

`unitMinSideWallCap`

A floating-point number representing the minimum capacitance.

`unitNomSideWallCap`

A floating-point number representing the nominal capacitance.

`unitMaxSideWallCap`

A floating-point number representing the maximum capacitance.

If you specify a zero (0), IC Compiler calculates sidewall capacitance based on the height from the substrate and the thickness of the layer, as specified in the physical attributes. (See “Physical Attributes” on page 1-32.)

Routing Channel Capacitance

Routing channel capacitance is defined per square user unit of a poly or metal layer in a routing channel.

The routing channel capacitance attributes are

`unitMinChannelCap`

A floating-point number representing the minimum capacitance.

`unitNomChannelCap`

A floating-point number representing the nominal capacitance.

`unitMaxChannelCap`

A floating-point number representing the maximum capacitance.

Total Sidewall Routing Channel Capacitance

Total sidewall routing channel capacitance is defined as the total sidewall capacitance per unit length for both sides of a poly or metal layer in a routing channel.

The sidewall routing channel capacitance specifications are

`unitMinChannelSideCap`

A floating-point number representing the minimum capacitance.

`unitNomChannelSideCap`

A floating-point number representing the nominal capacitance.

`unitMaxChannelSideCap`

A floating-point number representing the maximum capacitance.

Maximum Current Density

Use the `maxCurrDensity` attribute to define the maximum current density in amperes per centimeter for a layer.

Maximum Intracapacitance Distance Ratio

Use the `maxIntraCapDistRatio` attribute to specify the distance ratio required to control the extraction range. This attribute is a unitless floating-point number.

For example, where three times the `minSpacing` is required to control the extraction, specify a value of 3 for `maxIntraCapDistRatio`.

Wire Segment Length

Use the `maxSegLenForRC` attribute to define the maximum length in user units of a wire segment on the layer.

During extraction, IC Compiler splits wire segments longer than the specified length into segments of the specified length or less to compute a more accurate model. If you specify zero (0), the wire segments on the layer will not be split.

Physical Attributes

The physical attributes define physical characteristics of the layer and must be specified if you are doing timing-driven layout. This section describes the following physical attributes:

- Height From Substrate
- Thickness

Height From Substrate

The height from the substrate is the distance in user units between the layer and the substrate.

The height attributes are

`unitMinHeightFromSub`

A floating-point number representing the minimum distance.

`unitNomHeightFromSub`

A floating-point number representing the nominal distance.

`unitMaxHeightFromSub`

A floating-point number representing the maximum distance.

These values are used for calculating the sidewall capacitance when you do not specify sidewall capacitance. (See “Parasitic Attributes” on page 1-29.)

The height from substrate values are stored internally as a double-precision number and therefore is not limited by the database per-user unit limitations described in “Unit Precision and Range” on page 1-6.

Thickness

Thickness is the vertical thickness in user units of a poly or metal layer.

The thickness attributes are

`unitMinThickness`

A floating-point number representing the minimum thickness.

`unitNomThickness`

A floating-point number representing the nominal thickness.

`unitMaxThickness`

A floating-point number representing the maximum thickness.

This value is stored internally as a double-precision number and therefore is not limited by the database per-user unit limitations described in "Unit Precision and Range" on page 1-6.

Design Rule Attributes

The design rule attributes in a `Layer` section define layer-specific design rules; they apply only to the associated layer. All measurements in this section are in the user units specified in the Technology section of the technology file. (See "Technology Section" on page 1-5.) For more information about these design rules, see Chapter 2, "Routing Design Rules."

Note:

Interlayer design rules are defined in the `DesignRule` section. For more information, see "DesignRule Section" on page 1-47.

The design rule attributes for the `Layer` section are

`maxWidth`

Specifies the value used to identify wide metals for slotting and allows the `verify_drc` command to check the result of issuing the `slot_wires` command.

Note:

The signal router does not recognize the `maxWidth` rule.

`minWidth`

Defines the minimum width of any dimension of an object on the layer.

`defaultWidth`

Defines the default width of any dimension of an object on the layer.

The default width is used with all operations except the design rule checker (DRC), which uses the minimum width.

`minLength`

Defines the minimum wire length allowed on the layer.

`maxLength`

Defines the maximum length of an object (rectangle or polygon) on the layer.

If this attribute is set to 0, this rule is unspecified on this layer. Therefore, the router will not do any checking.

`minArea`

The minimum area for any object on the layer.

`specialMinArea`

The minimum area for a special shape in the two-stage minimum area rule.

`minAreaEdgeThreshold`

The edge length threshold used to define special shapes for the two-stage minimum area rule.

`minEnclosedArea`

The minimum area enclosed by ring-shaped wires or vias.

`minEnclosedWidth`

The minimum width of any dimension of an enclosed area on the layer.

`minEdgeLength`

The maximum threshold for short edges.

`maxNumMinEdge`

The maximum number of consecutive short edges.

`maxTotalMinEdgeLength`

The maximum total length of consecutive edges shorter than `minEdgeLength`.

`minEdgeLength2`

The length threshold for applying the special notch rule. Use with `minEdgeLength3`.

`minEdgeLength3`

The minimum width of the notch for the special notch rule. Use with `minEdgeLength2`.

`minSpacing`

The minimum spacing between the edges of objects on the layer.

`cornerMinSpacing`

The minimum corner-to-corner spacing between two vias. This value is smaller than `minSpacing`.

`sameNetMinSpacing`

The minimum spacing for two shapes belonging to the same net. This value is smaller than `minSpacing`.

`checkManhattanSpacing`

Specifies whether diagonal or rectilinear distance measurement is used for DRC violation analysis. If this attribute is set to 0, diagonal measurement is used; the actual spacing, measured diagonally, must satisfy the minimum spacing requirement. If it is set to 1, Manhattan (rectilinear) measurement is used; at least one of the X and Y directions must satisfy the minimum spacing requirement.

`cutTblSize`

The size of the cut table for the general cut spacing rule.

`cutNameTbl`

A list of names assigned to different cut sizes for the general cut spacing rule.

`cutWidthTbl`

The list of cut widths corresponding to the list of names in the `cutNameTbl`.

`cutHeightTbl`

The list of cut heights corresponding to the list of names in the `cutNameTbl`.

`cutDataTypesTbl`

The list of data types corresponding to the list of names in the `cutNameTbl`.

`maxNumAdjacentCut`

The maximum number of adjacent vias.

`adjacentCutRange`

The distance for defining adjacent vias.

`enclosedCutNumNeighbor`

The minimum number of neighboring vias allowed for defining an enclosed via.

`enclosedCutNeighborRange`

The distance range of neighboring vias for defining an enclosed via.

`enclosedCutMinSpacing`

The minimum spacing between two enclosed vias.

`enclosedCutToNeighborMinSpacing`

The minimum spacing between an enclosed via and its neighboring vias.

`onGrid`

Specifies whether the router forces vias to be placed on the grid.

`endOfLine1NeighborThreshold`

The line width threshold that determines the application of the end-of-line to end-of-line and one-neighbor end-of-line spacing rules.

`endOfLine1NeighborEndToEndMinSpacing`

The minimum spacing between the two line-ends in the end-of-line to end-of-line spacing rule.

`endOfLine1NeighborEndToEndCornerKeepoutWidth`

The extension of the keepout area beyond the line-end corners in the end-of-line to end-of-line spacing rule.

`endOfLine1NeighborMinSpacing`

The minimum spacing between the line-end and the neighboring metal in the one-neighbor end-of-line spacing rule.

`endOfLine1NeighborCornerKeepoutWidth`

The extension of the keepout area beyond the line-end corners in the one-neighbor end-of-line spacing rule.

`endOfLine2NeighborThreshold`

The line width threshold that determines the application of the two-neighbor end-of-line spacing rule.

`endOfLine2NeighborMinSpacing`

The minimum spacing between the line-end and the neighboring metal in the two-neighbor end-of-line spacing rule.

`endOfLine2NeighborSideMinSpacing`

The minimum spacing between the side of the line-end and the neighboring metal in the two-neighbor end-of-line spacing rule.

`endOfLine2NeighborCornerKeepoutWidth`

The extension of the keepout area beyond the line-end corners in the two-neighbor end-of-line spacing rule.

`endOfLine2NeighborSideKeepoutLength`

The length of the keepout area along the side of the line-end metal in the two-neighbor end-of-line spacing rule.

`endOfLine3NeighborThreshold`

The line width threshold that determines the application of the three-neighbor end-of-line spacing rule.

`endOfLine3NeighborMinSpacing`

The minimum spacing between the line-end and the neighboring metal in the three-neighbor end-of-line spacing rule.

`endOfLine3NeighborSideMinSpacing`

The minimum spacing between the sides of the line-end and the two side-neighboring metals in the three-neighbor end-of-line spacing rule.

`endOfLine3NeighborCornerKeepoutWidth`

The extension of the keepout area beyond the line-end corners in the three-neighbor end-of-line spacing rule.

`endOfLine3NeighborSideKeepoutLength`

The length of the keepout areas along the sides of the line-end metal in the three-neighbor end-of-line spacing rule.

`stubThreshold`

The maximum parallel length for applying the `stubSpacing` rule instead of the `minSpacing` rule.

`stubLengthThreshold`

The length threshold of wire edges that share the same corners with the end-of-line edge. `stubSpacing` is required when the end-of-line edge's width is less than or equal to `stubThreshold` and either adjacent edge's length is less than or equal to `stubLengthThreshold`.

`stubSpacing`

The minimum spacing for two objects that run parallel for less than the distance specified in `stubThreshold`. This value is smaller than `minSpacing`.

`stubToStubSpacing`

The spacing between two end-of-line wire segments.

Note:
The `stubToStubSpacing` rule is not supported by Zroute.

`endOfLineCornerKeepoutWidth`

The distance from the corner of the end-of-line metal segment to the neighboring metal segment, defining a keepout region.

`stubMinLength`

The minimum distance from the end-of-line edge to the opposite internal edge.

`tJunctionStubWireMaxThreshold`

The end-of-line wire width threshold for applying the two-sided end-of-line spacing rule.

`tJunctionOrthoWireMaxThreshold`

The nearby-wire width threshold for applying the two-sided end-of-line spacing rule.

`tJunctionStubKeepoutMinSpacing`

The width of the keepout area, perpendicular to the wire end, for the two-sided end-of-line spacing rule.

`tJunctionStubKeepoutMinWidth`

The length of the keepout area, from wire end to wire end, for the two-sided end-of-line spacing rule.

`uShapeMinSpacing`

The minimum spacing between two edges when at least one edge has a U-shaped notch.

`uShapeDepthThreshold`

The minimum depth threshold of the notch for applying the U-shape minimum spacing rule.

`uShapeMinLength`

The maximum length threshold of the notch for applying the U-shape minimum spacing rule.

`sameNetWidthThreshold`

The width threshold for the thin wire width for the dog-bone rule.

`maxStackLevel`

The maximum number of vias that can stack at the same point.

`fatWireThreshold`

The threshold for using the `fatFatMinSpacing` rule instead of the `minSpacing` rule.

`fatFatMinSpacing`

The minimum distance required between wires on a layer when the widths of both wires are greater than or equal to `fatWireThreshold`.

`fatThinMinSpacing`

The minimum distance required between wires on a layer when the width of one of the wires is greater than or equal to `fatWireThreshold`.

`fatWireExtensionRange`

The extension range required for using the fat wire spacing rules (`fatFatMinSpacing` or `fatThinMinSpacing`) instead of the `minSpacing`, even when the wire in the extension portion is no longer fat. Any metal extending out from the fat wire must be treated as fat within the specified extension range.

`fatTblDimension`

The size of the fat table.

`fatTblThreshold`, `fatTblThreshold2`

The thresholds used to define the different fat degrees of a metal segment. You must specify *n* values, where *n* is the table size.

`fatTblParallelLength`

The parallel length thresholds used to adjust the corresponding fat degree of a metal when the metal segment is adjacent to another metal segment. You must specify *n* values, where *n* is the table size.

`fatTblExtensionRange`

The extension ranges required for a metal segment of the corresponding fat degree.

`fatTblSpacing`

The spacing required for the corresponding fat condition when two metal segments are adjacent. You must specify an *n* by *n* table, where *n* is the table size.

`fatTblMinEnclosedArea`

The minimum enclosed area for the fat wires. You must specify *n* values, where *n* is the table size.

`fatTblXDimension`

The size of the fat metal orthogonal X spacing rule table.

`fatTblYDimension`

The size of the fat metal orthogonal Y spacing rule table.

`fatTblXThreshold`

The thresholds used to define the different fat degrees of a metal segment in the X direction. You must specify *n* values, where *n* is the X table size.

`fatTblYThreshold`

The thresholds used to define the different fat degrees of a metal segment in the Y direction. You must specify *n* values, where *n* is the Y table size.

`fatTblXParallelLength`

The parallel length thresholds used to adjust the corresponding fat degree of a metal when the metal segment is adjacent to another metal segment, for the X spacing rule. You must specify *n* values, where *n* is the X table size.

`fatTblYParallelLength`

The parallel length thresholds used to adjust the corresponding fat degree of a metal when the metal segment is adjacent to another metal segment, for the Y spacing rule. You must specify *n* values, where *n* is the Y table size.

`fatTblXMinSpacing`

The minimum X spacing required for the corresponding fat condition when two metal segments are adjacent. You must specify an *n* by *n* table, where *n* is the table size.

`fatTblYMinSpacing`

The minimum Y spacing required for the corresponding fat condition when two metal segments are adjacent. You must specify an *n* by *n* table, where *n* is the table size.

`orthoSpacingExcludeCorner`

Specifies whether to extend the spacing check to the corner area for the given layer for the fat metal orthogonal spacing rule.

`fatContactThreshold`

The threshold for using a fat wire contact instead of the default contact.

Any wire on the specified layer whose width is equal to or greater than this threshold requires a fat wire contact. For more information, see the description of `isFatContact` in "ContactCode Section" on page 1-41.

`fatTblFatContactNumber`

The contact code numbers for the fat wires in a one-dimensional table rule. You must specify *n* values, where *n* is the table size.

`fatTblFatContactMinCuts`

The minimum number of vias in a one-dimensional table rule. You must specify *n* values, where *n* is the table size.

`fat2DTblFatContactNumber`

The contact code numbers for the fat wires in a two-dimensional table rule. You must specify an *n* by *n* table, where *n* is the table size.

`fat2DTblFatContactMinCuts`

The minimum number of vias in a two-dimensional table rule. You must specify an *n* by *n* table, where *n* is the table size.

`fatTblExtensionContactNumber`

The contact code numbers for the fat wires within the extension range of fat wires. You must specify *n* values, where *n* is the table dimension.

`fatTblExtensionMinCuts`

The minimum number of vias for the wires within the extension range of fat wires. You must specify *n* values, where *n* is the table dimension.

`spanTblDimension`

The table size for the metal span spacing rule.

`spanTblThreshold`

A table of metal span values for the metal span spacing rule.

`spanTblParallelLength`

A table of parallel length values for the metal span spacing rule

`spanTblMinSpacing`

A table of minimum spacing values corresponding to the combinations of span thresholds and parallel lengths for the metal span spacing rule.

`protrusionTblDim`

The size of the protrusion table.

`protrusionFatThresholdTbl`

The threshold value (minimum widths) for the fat wire. You must specify *n* values, where *n* is the table size.

`protrusionLengthLimitTbl`

The length threshold (maximum lengths) for the connected thin wire. You must specify *n* values, where *n* is the table size.

`protrusionMinWidthTbl`

The minimum width value for the connected thin wire. You must specify *n* values, where *n* is the table size.

ContactCode Section

A technology file defines each via master (contact code) by specifying attributes for that via master. A via master can be a default via master (default contact) or a fat via master (fat contact). The attributes fall into several groupings, including

- Physical attributes

  Physical attributes define the physical attributes of the via master.

- Parasitic attributes

  Parasitic attributes define the layer parasitics for a via layer.

- Design rule attributes

  Design rule attributes define the layer-specific design rules associated with the via layer.

The `ContactCode` section defines the via masters (contact codes) used in designs in the library.

Example 1-10 shows a typical `ContactCode` section for a default via.

Example 1-10 ContactCode Section (DefaultContact)

```
ContactCode "PCON" {
      /* physical attributes */
      contactCodeNumber = 1
      contactSourceType = 0
      cutLayer = "CONT"
      lowerLayer = "M1"
      upperLayer = "M2"
      isDefaultContact = 1
      cutWidth = 0.8
      cutHeight = 0.8

      /* parastic attributes */
      unitMinResistance = 0.00025
      unitNomResistance = 0.00025
      unitMaxResistance = 0.00025
      temperatureCoeff = 2.5e-06
      unitMinCapacitance = 0.0004
      unitNomCapacitance = 0.0004
      unitMaxCapacitance = 0.0004

      /* design rule attributes */
      upperLayerEncWidth = 0.6
      upperLayerEncHeight = 0.6
      lowerLayerEncWidth = 0.6
      lowerLayerEncHeight = 0.6
      minCutSpacing = 1.2
      maxNumRowsNonTurning = 1

}
```

Example 1-11 shows a typical `ContactCode` section for a fat via.

Example 1-11 ContactCode Section (FatContact)

```
ContactCode "PCON" {
      /* physical attributes */
      contactCodeNumber = 1
      contactSourceType = 0
      cutLayer = "CONT"
      lowerLayer = "M1"
      upperLayer = "M2"
      isFatContact = 1
      cutWidth = 0.8
      cutHeight = 0.8
```

```
      /* parastic attributes */
      unitMinResistance = 0.00025
      unitNomResistance = 0.00025
      unitMaxResistance = 0.00025
      unitMinCapacitance = 0.0004
      unitNomCapacitance = 0.0004
      unitMaxCapacitance = 0.0004

      /* design rule attributes */
      upperLayerEncWidth = 0.6
      upperLayerEncHeight = 0.6
      lowerLayerEncWidth = 0.6
      lowerLayerEncHeight = 0.6
      minCutSpacing = 1.2
      maxNumRowsNonTurning = 1
}
```

Physical Attributes

The physical attributes define the physical characteristics of the via master.

The physical attributes are

name

The name of the via master.

`contactCodeNumber`

Identifies the via master. This must be an integer between 1 and 255. Each `ContactCode` section must have a unique `contactCodeNumber`.

`contactSourceType`

Identifies the contact type. Valid values and their meanings are

- 0 (single-cut fixed via - this is the default contact type)
- 1 (generated via for regular nets)
- 2 (multiple-cut fixed via)
- 3 (single-cut via for nondefault routing rule)
- 4 (multiple-cut via for nondefault routing rule)
- 5 (generated via for special nets, such as power nets)

`cutLayer`

Specifies the name of the display layer used for the via. The value must be the name of one of the layers specified in a `Layer` section.

`lowerLayer`

Specifies the name of the display layer used for one of the conduction layers. The value must be the name of one of the layers specified in a `Layer` section.

`upperLayer`

Specifies the name of the display layer used for the other conduction layer. The value must be the name of one of the layers specified in a `Layer` section.

`isDefaultContact`

Specifies whether the contact is a default contact. Valid values are 0 or 1.

Note:

A given `cutLayer` must have no more than one default contact. If no default contact is specified, the contact with the lowest number is the default.

`isFatContact`

Specifies whether the contact is a fat contact. Valid values are 0 or 1.

A fat contact is used when the wire is wider than the `fatContactThreshold` value specified in the associated `Layer` section.

`cutHeight`

Specifies the vertical dimension of the cut.

`cutWidth`

Specifies the horizontal dimension of the cut.

Parasitic Attributes

The parasitic attributes define the layer parasitics. In general, IC Compiler gets the parasitic information from the TLUPlus files, rather than the technology file.

This section describes the following parasitic attributes:

- Resistance
- Capacitance
- Maximum Current Density

Resistance

Resistance is defined as the resistance per contact. The resistance attributes are

`unitMinResistance`

A floating-point number representing the minimum resistance.

`unitNomResistance`

A floating-point number representing the nominal resistance.

`unitMaxResistance`

A floating-point number representing the maximum resistance.

Capacitance

Capacitance is defined per contact in a cell instance or over a macro. The capacitance attributes are

`unitMinCapacitance`

A floating-point number representing the minimum capacitance.

`unitNomCapacitance`

A floating-point number representing the nominal capacitance.

`unitMaxCapacitance`

A floating-point number representing the maximum capacitance.

Maximum Current Density

Use the `maxCurrDensity` attribute to define the maximum current density in amperes per centimeter that the via can carry.

Design Rule Attributes

The design rule attributes in a `ContactCode` section define design rules specific to a via layer. All measurements in this section are in the user units specified in the Technology section of the technology file. (See "Technology Section" on page 1-5.)

Note:

Additional via design rules are defined in the `Layer` and `DesignRule` sections. For more information, see "Design Rule Attributes" on page 1-33 and "DesignRule Section" on page 1-47.

The design rule attributes for the `ContactCode` section are

`upperLayerEncHeight`

The distance at which the first of two layers encloses the via in the vertical dimension.

`upperLayerEncWidth`

The distance at which the first of two layers encloses the via in the horizontal dimension.

`lowerLayerEncHeight`

The distance at which the second of two layers encloses the via in the vertical dimension.

`lowerLayerEncWidth`

The distance at which the second of two layers encloses the via in the horizontal dimension.

`minCutSpacing`

The minimum separation distance between the edges of the via.

`viaFarmSpacing`

The minimum separation distance between two via farms.

`maxNumRows`

When `viaFarmSpacing` is defined in the same `ContactCode` section, the `maxNumRows` value represents the exact number of rows in each via farm in a via farm array; otherwise, the `maxNumRows` value represents the maximum number of rows in a via array.

`maxNumRowsNonTurning`

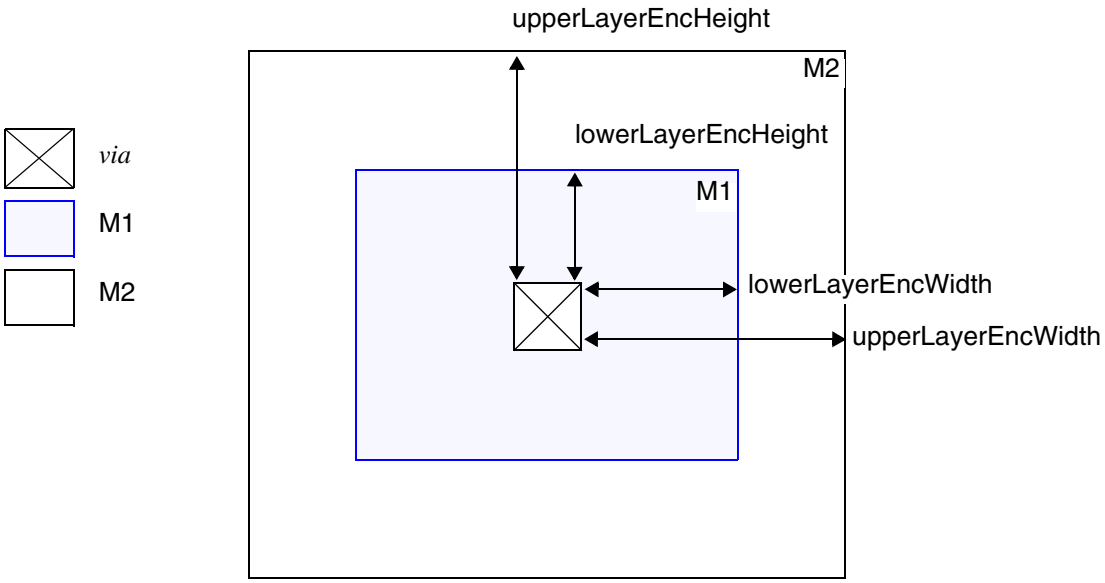
Restricts a via size by placing an upper limit on the number of rows of cuts allowed in the routing direction.

Note:

When the prerouter needs to switch layers without changing the routing direction, it uses a square for a drop via. The width and height of the square is the fixed width of the wires.

Figure 1-2 shows the interpretation of the `upperLayerEncHeight`, `lowerLayerEncHeight`, `upperLayerEncWidth`, and `lowerLayerEncWidth` attributes when the lower layer is M1 and the upper layer is M2.

Figure 1-2 Via Enclosure Attributes



DesignRule Section

The `DesignRule` section of the technology file defines the interlayer design rules that apply to designs in the library. All measurements in this section are in the user units specified in the Technology section of the technology file. (See "Technology Section" on page 1-5.) For more information about these design rules, see Chapter 2, "Routing Design Rules."

Note:

Layer-specific design rules are defined in a `Layer` section. For more information, see "Design Rule Attributes" on page 1-33.

Example 1-12 shows all the attributes for a `DesignRule` section.

Example 1-12 DesignRule Section

```
DesignRule {

      layer1 = "name"
      layer2 = "name"
      minSpacing = 1.0
      diffNetMinSpacing = 1.0
      cornerMinSpacing = 0
      minEnclosure = 0
      cut1TblSize = 0
      cut2TblSize = 0
      cut1NameTbl = (name1, name2)
```

```
        cut2NameTbl = (name1, name2)
        orthoSpacingExcludeCornerTbl = (0, 0, 0, 0)
        sameNetXMinSpacingTbl = (0, 0, 0, 0)
        sameNetYMinSpacingTbl = (0, 0, 0, 0)
        sameNetCornerMinSpacingTbl = (0, 0, 0, 0)
        sameNetCenterMinSpacingTbl = (0, 0, 0, 0)
        diffNetXMinSpacingTbl = (0, 0, 0, 0)
        diffNetYMinSpacingTbl = (0, 0, 0, 0)
        diffNetCornerMinSpacingTbl= (0, 0, 0, 0)
        diffNetCenterMinSpacingTbl= (0, 0, 0, 0)
        sameSegXMinSpacingTbl = (0, 0, 0, 0)
        sameSegYMinSpacingTbl = (0, 0, 0, 0)
        sameSegCornerMinSpacingTbl = (0, 0, 0, 0)
        sameSegCenterMinSpacingTbl = (0, 0, 0, 0)
        diffSegXMinSpacingTbl = (0, 0, 0, 0)
        diffSegYMinSpacingTbl = (0, 0, 0, 0)
        diffSegCornerMinSpacingTbl = (0, 0, 0, 0)
        diffSegCenterMinSpacingTbl = (0, 0, 0, 0)
        endOfLineEnclosure = 0
        endOfLineViaJogLenth = 0
        endOfLineViaJogWidth = 0
        endOfLineViaEncWidth = 0
        stackable = 0
        endOfLineEncTblSize = 0
        endOfLineEncTbl = (0, 0, 0, 0, 0, 0, 0, 0)
        endOfLineEncSideThreshold = (0, 0, 0, 0, 0, 0, 0, 0)
        endOfLineEncSpacingTbl = (0, 0, 0, 0, 0, 0, 0, 0)
        endOfLineTShapeEncTblSize = 0
        endOfLineTShapeCornerMinSpacing = (0, 0, 0, 0, 0, 0, 0, 0)
        endOfLineTShapeEncSideThreshold = (0, 0, 0, 0, 0, 0, 0, 0)
        endOfLineTShapeEncTbl = (0, 0, 0, 0, 0, 0, 0, 0)
        endOfLineTShapeEncSpacingTbl = (0, 0, 0, 0, 0, 0, 0, 0)
        endOfLineEnc2NeighborTblSize = 0
        endOfLineEnc2NeighborThreshold = 0
        endOfLineEnc2NeighborCornerKeepoutWidth = 0
        endOfLineEnc2NeighborSideKeepoutLength = 0
        endOfLineEnc2NeighborSideMinSpacing = 0
        endOfLineEnc2NeighborMinEnclosure = 0
        endOfLineEnc2NeighborTbl = (0, 0, 0)
        endOfLineEnc2NeighborSpacingTbl = (0, 0, 0)
        endOfLineEnc2NeighborViaArrayExcludedTbl=(0, 0, 0)
        endOfLineEnc2NeighborWireMinThreshold = 0
        jogWireViaKeepoutTblSize = 0
        jogWireViaKeepoutEncThreshold = (0, 0, 0, 0, 0, 0, 0, 0)
        jogWireViaKeepoutMinSize = (0, 0, 0, 0, 0, 0, 0, 0)
        fatWireViaKeepoutTblSize = 0
        fatWireViaKeepoutWidthThreshold = (0, 0, 0, 0, 0, 0, 0, 0)
        fatWireViaKeepoutParallelLengthThreshold = (0, 0, 0, 0, 0, 0, 0, 0)
        fatWireViaKeepoutMaxSpacingThreshold = (0, 0, 0, 0, 0, 0, 0, 0)
        fatWireViaKeepoutMinSize = (0, 0, 0, 0, 0, 0, 0, 0)
        fatWireViaKeepoutEnclosure = (0, 0, 0, 0, 0, 0, 0, 0)
}
```

The attribute definitions are:

`layer1`

One of the layers to which the rule applies. The value must be the name of one of the layers specified in a `Layer` section.

`layer2`

The other layer to which the rule applies. The value must be the name of one of the layers specified in a `Layer` section.

`minSpacing`

The minimum spacing between the specified layers.

`diffNetMinSpacing`

The minimum different-net cut-to-cut spacing for the specified layers.

`cornerMinSpacing`

The minimum corner-to-corner spacing between two vias on different via layers. This value is smaller than `minSpacing`.

`minEnclosure`

The minimum distance at which a layer must enclose another layer when the two layers overlap.

`cut1TblSize`

The size of the cut 1 name table used in the general cut spacing rule.

`cut2TblSize`

The size of the cut 2 name table used in the general cut spacing rule.

`cut1NameTbl`

A list of cut names previously defined by the `cutNameTbl` attribute used in the general cut spacing rule.

`cut2NameTbl`

A list of cut names previously defined by the `cutNameTbl` attribute used in the general cut spacing rule.

`orthoSpacingExcludeCornerTbl`

Specifies whether the general cut spacing rule is extended beyond the corners of the cut.

`sameNetXMinSpacingTbl`

The minimum X spacing between cuts in the same net for the general cut spacing rule.

`sameNetYMinSpacingTbl`

The minimum Y spacing between cuts in the same net for the general cut spacing rule.

`sameNetCornerMinSpacingTbl`

The minimum corner-to-corner spacing between cuts in the same net for the general cut spacing rule.

`sameNetCenterMinSpacingTbl`

The minimum center-to-center spacing between cuts in the same net for the general cut spacing rule.

`diffNetXMinSpacingTbl`

The minimum X spacing between cuts in different nets for the general cut spacing rule.

`diffNetYMinSpacingTbl`

The minimum Y spacing between cuts in different nets for the general cut spacing rule.

`diffNetCornerMinSpacingTbl`

The minimum corner-to-corner spacing between cuts in different nets for the general cut spacing rule.

`diffNetCenterMinSpacingTbl`

The minimum center-to-center spacing between cuts in different nets for the general cut spacing rule.

`sameSegXMinSpacingTbl`

The minimum X spacing between cuts in the same wire segment for the general cut spacing rule.

`sameSegYMinSpacingTbl`

The minimum Y spacing between cuts in the same wire segment for the general cut spacing rule.

`sameSegCornerMinSpacingTbl`

The minimum corner-to-corner spacing between cuts in the same wire segment for the general cut spacing rule.

`sameSegCenterMinSpacingTbl`

The minimum center-to-center spacing between cuts in the same wire segment for the general cut spacing rule.

`diffSegXMinSpacingTbl`

The minimum X spacing between cuts in different wire segments for the general cut spacing rule.

`diffSegYMinSpacingTbl`

The minimum Y spacing between cuts in different wire segments for the general cut spacing rule.

`diffSegCornerMinSpacingTbl`

The minimum corner-to-corner spacing between cuts in different wire segments for the general cut spacing rule.

`diffSegCenterMinSpacingTbl`

The minimum center-to-center spacing between cuts in different wire segments for the general cut spacing rule.

`endOfLineEnclosure`

An enclosure size to specify the end-of-line rule for routing wire segments.

`endOfLineViaJogLength`

The length from the jog wire edge to the end-of-line edge, which is also the minimum spacing between the end-of-line edge and a single via.

`endOfLineViaJogWidth`

The maximum width of the end-of-line edge.

`endOfLineViaEncWidth`

The maximum width of the single-via enclosure.

`stackable`

Controls whether two vias can be stacked if their cuts are on layer1 and layer2. Valid values are 0 or 1.

When no spacing rule exists between the two layers, the `stackable` attribute does not need to be specified and the router allows contacts to overlap arbitrarily.

When a spacing rule exists between the two layers and `stackable` is set to 1, the router does one of the following:

- Separates the contacts with the spacing rule
- Stacks the two contacts, aligning the centers

`endOfLineEncTblSize`

The size of the end-of-line enclosure table.

`endOfLineEncTbl`

The minimum metal enclosure required at the end-of-line edge. You must specify *n* values, where *n* is the table size.

`endOfLineEncSideThreshold`

The metal enclosure thresholds at the side edges of an end-of-line wire that encloses a via near the end-of-line edge. You must specify *n* values, where *n* is the table size.

`endOfLineEncSpacingTbl`

The minimum end-of-line spacing for wire ends with a via dropped to a lower via layer when the width of the wire is less than `stubThreshold`. You must specify *n* values, where *n* is the table size.

`endOfLineTShapeEncTblSize`

The size of the T-shape end-of-line enclosure table.

`endOfLineTShapeCornerMinSpacing`

The maximum spacing threshold between the corner of the T-shape wire and the corner of the enclosed via for applying the T-shape wire via enclosure rule. If the spacing is greater than this value, the rule is waived.

`endOfLineTShapeEncSideThreshold`

The width threshold between the edge of the via and the edge of the enclosing T-shape wire end. You must specify *n* values, where *n* is the table size.

`endOfLineTShapeEncTbl`

The minimum extension between a single via's edge and the edges of the enclosing T-shape wire. You must specify *n* values, where *n* is the table size.

`endOfLineTShapeEncSpacingTbl`

The minimum end-of-line spacing for T-shape wire ends with a via dropped to a lower via layer based on the wire width ranges specified in the `endOfLineTShapeEncSideThreshold` attribute. You must specify *n* values, where *n* is the table size.

`jogWireViaKeepoutTblSize`

The size of the jog wire via keepout table.

`endOfLineEnc2NeighborTblSize`

The table size for the two-neighbor end-of-line via enclosure rule.

`endOfLineEnc2NeighborThreshold`

The line-width threshold for applying the two-neighbor end-of-line via enclosure rule.

`endOfLineEnc2NeighborCornerKeepoutWidth`

The extension of the keepout area beyond the line-end corners in the two-neighbor end-of-line via enclosure rule.

`endOfLineEnc2NeighborSideKeepoutLength`

The length of the keepout area along the side of the line-end metal in the two-neighbor end-of-line via enclosure rule.

`endOfLineEnc2NeighborSideMinSpacing`

The minimum spacing between the side of the line-end and the neighboring metal in the two-neighbor end-of-line via enclosure rule.

`endOfLineEnc2NeighborMinEnclosure`

The minimum overlap of a line-end over the via in the two-neighbor end-of-line via enclosure rule.

`endOfLineEnc2NeighborTbl`

A table of minimum line-end enclosure values in the two-neighbor end-of-line via enclosure rule.

`endOfLineEnc2NeighborSpacingTbl`

A table of line-end to neighboring-metal spacing values in the two-neighbor end-of-line via enclosure rule.

`endOfLineEnc2NeighborViaArrayExlcudeTbl`

A table of values, either 0 or 1, used to not exclude or exclude the application of the rule to via arrays.

`endOfLineEnc2NeighborWireMinThreshold`

The minimum edge length for the two-neighbor end-of-line via enclosure rule to be applied.

`jogWireViaKeepoutEncThreshold`

The maximum width thresholds for applying the jog wire via keepout rule. You must specify *n* values, where *n* is the table size.

`jogWireViaKeepoutMinSize`

The minimum spacing between the edge of the via and the corner edges of a jog wire.

`fatWireViaKeepoutTblSize`

The size of the fat wire via keepout table.

`fatWireViaKeepoutWidthThreshold`

The minimum width thresholds of the enclosing metal segment. You must specify *n* values, where *n* is the table size.

`fatWireViaKeepoutParallelLengthThreshold`

The minimum parallel length thresholds between the enclosing metal segment and the neighboring metal segment. You must specify *n* values, where *n* is the table size.

`fatWireViaKeepoutMaxSpacingThreshold`

The maximum spacing thresholds between the enclosing metal segment and the neighboring metal segment. You must specify *n* values, where *n* is the table size.

`fatWireViaKeepoutMinSize`

The minimum required length of the keepout area in each direction from the corner. You must specify *n* values, where *n* is the table size.

`fatWireViaKeepoutEnclosure`

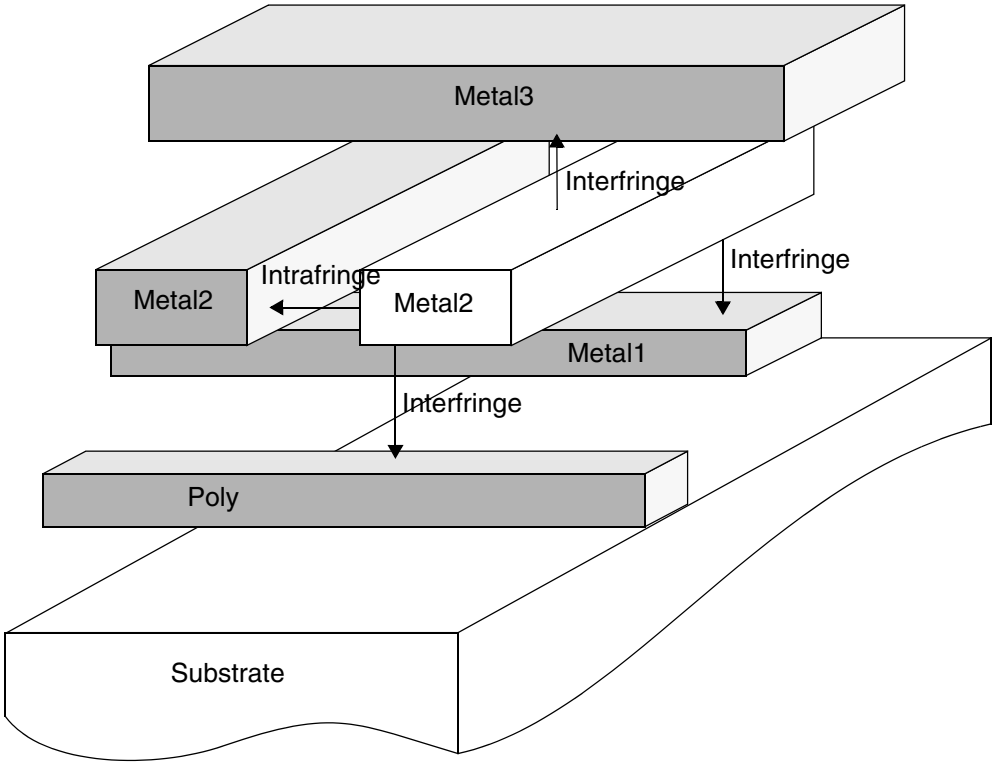
The minimum required width of the metal enclosure. You must specify *n* values, where *n* is the table size.

FringeCap Section

A `FringeCap` section specifies capacitance information for interconnect layers when they are overlapping or parallel to each other. This information includes the following:

- Capacitance per unit area when objects on different layers overlap (interfringe in Figure 1-3)
- Capacitance per unit length when objects on the same layer are separated by the minimum spacing specified in the `Layer` section (intrafringe in Figure 1-3)

Figure 1-3 Interfringe and Intrafringe Examples



Default values for coupling capacitance between adjoining parallel objects on the same layer or adjoining objects on different layers are defined in the respective `Layer` sections of the technology file. Other coupling capacitance values are entered in the `FringeCap` section.This information is used for calculating coupling capacitance for timing-driven layout.

Note:
In general, IC Compiler gets the parasitic information from the TLUPlus files, rather than the technology file.

The attributes in a `FringeCap` section are

`number`

The coupling capacitance number, which must be an integer value at least one less than the number of metal layers.

`layer1`

One of the two metal layers on which different metal layers cross each other.

`layer2`

The other metal layer on which different metal layers cross each other.

`minFringeCap`

The minimum capacitance for the layer.

`nomFringeCap`

The nominal capacitance for the layer.

`maxFringeCap`

The maximum capacitance for the layer.

Example 1-13 shows a typical interfringe `FringeCap` section.

Example 1-13 FringeCap Section (Interfringe)

```
FringeCap 4 {
      number = 4
      layer1 = "M2"
      layer2 = "M3"
      minFringeCap = 0.00022
      nomFringeCap = 0.00022
      maxFringeCap = 0.00022
}
```

Example 1-14 shows a typical intrafringe FringeCap section.

Example 1-14 FringeCap Section (Intrafringe)

```
FringeCap 4 {
      number = 4
      layer1 = "M2"
      layer2 = "M2"
      minFringeCap = 0.00017
      nomFringeCap = 0.00017
      maxFringeCap = 0.00017
}
```

CapModel and CapTable Sections

The `CapModel` and `CapTable` sections of a technology file let you specify the material type (conductor or dielectric), thickness, and dielectric value for each layer, as well as the wire width and spacing for each layer.

Note:

In general, IC Compiler gets the parasitic information from the TLUPlus files, rather than the technology file.

You can create these two sections manually with a text editor, but you normally create them by using the Milkyway Environment `cmCreateCapModel` command. For more information about this command, see the *IC Compiler Data Preparation Using Milkyway User Guide*.

Example 1-15 is an example of a `CapModel` section.

Example 1-15 CapModel Section

```
CapModel "polyConfig1" {
      refLayer = "poly"
      groundPlaneBelow = ""
      groundPlaneAbove = "metal1"
      bottomCapType = "Table"
      bottomCapDataMin = "poly_C_BOTTOM_GP"
      bottomCapDataNom = "poly_C_BOTTOM_GP"
      bottomCapDataMax = "poly_C_BOTTOM_GP"
      topCapType = "Table"
      topCapDataMin = "poly_C_TOP_GP"
      topCapDataNom = "poly_C_TOP_GP"
      topCapDataMax = "poly_C_TOP_GP"
      lateralCapType = "Table"
      lateralCapDataMin = "poly_C_LATERAL"
      lateralCapDataNom = "poly_C_LATERAL"
      lateralCapDataMax = "poly_C_LATERAL"
}
```

Example 1-16 is an example of a `CapTable` section.

Example 1-16 CapTable Section

```
CapTable "metal3_C_BOTTOM_GP_21" {
         wireWidthSize = 3
         wireSpacingSize = 5
         wireWidth = (0.6, 1.2, 1.8)
         wireSpacing = (0.5, 1, 1.5, 2, 2.5)
         capValue = (
                     2.4798e-05, 3.05934e-05, 3.5548e-05, 3.99643e-05,
                     4.39498e-05, 3.1404e-05, 3.74482e-05, 4.2877e-05,
                     4.77828e-05, 5.22164e-05, 3.87834e-05, 4.50642e-05,
                     5.07664e-05, 5.59294e-05, 6.05912e-05
                     )
}
```

Example 1-17 is an example of a table lookup capacitance model.

Example 1-17 Table Lookup Capacitance Model

```
Technology {
      name = ""
      dielectric = 0
}

Stipple "solid" {
      width = 1
      height = 1
      pattern = (1, 1, 1, 1)
}

CapTable "poly_C_TOP_GP" {
      wireWidthSize = 2
      wireSpacingSize = 3
      wireWidth = (0.4, 0.8)
      wireSpacing = (0.5, 1, 1.5)
      capValue = (4.68407e-05, 5.76469e-05,
                  6.10734e-05, 6.97226e-05,
                  8.06076e-05, 8.40257e-05)
}
......
```

ResModel Section

The `ResModel` section in the technology file allows the resistance and temperature coefficient of the layer to be expressed as a function of wire width.

Note:

In general, IC Compiler gets the parasitic information from the TLUPlus files, rather than the technology file.

When the `ResModel` section for a given layer is present, it overrides the values for the following attributes in the corresponding `Layer` section: `unitMinResistance`, `unitNomResistance`, and `unitMaxResistance`.

Example 1-18 shows a `ResModel` section.

Example 1-18 ResModel Section

```
ResModel "metal1ResModel" {
      layerNumber = 1
      size = 3
      wireWidth = (0.48, 0.6, 0.72)
      tempCoeff = (0.01, 0.01, 0.01)
      minRes = (0.0003, 0.0002, 0.0001)
      nomRes = (0.0003, 0.0002, 0.0001)
      maxRes = (0.0003, 0.0002, 0.0001)
}
```

PRRule Section

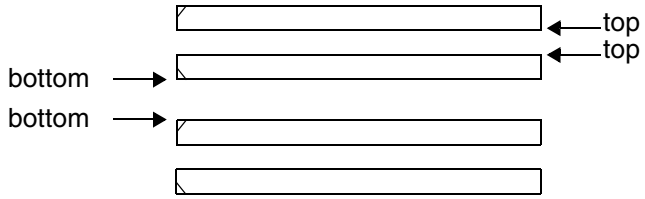
The `PRRule` section of a technology file defines cell row spacing. The cell row spacing rules differ depending on whether you are using double-back cell rows or not.

Cell Row Spacing Rules for Double-Back Cell Rows

When using double-back cell rows in your floorplan, as shown in Figure 1-4, you specify the following row spacing rules:

- Between top edge and top edge
- Between bottom edge and bottom edge

Figure 1-4 Row Spacing Rule for Double-Back Cells



The attributes used to define these rules are

`rowSpacingTopTop`

The spacing between the top edges when you are using double-back cell rows in your floorplan.

`rowSpacingBotBot`

The spacing between the bottom edges when you are using double-back cell rows in your floorplan.

`abuttableTopTop`

Whether or not the two top edges can be abutted when you are using double-back cell rows in your floorplan. Valid values are 1 or 0.

`abuttableBotBot`

Whether or not the two bottom edges can be abutted when you are using double-back cell rows in your floorplan. Valid values are 1 or 0.

Example 1-19 shows a typical `PRRule` section for double-back cell rows.

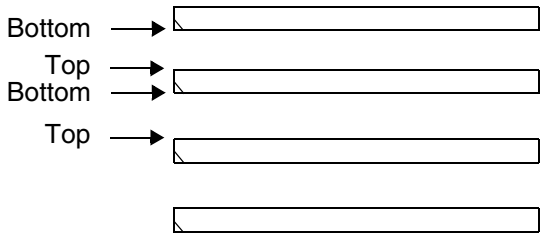
Example 1-19 PRRule Section for Double-Back Cell Rows

```
PRRule {
      rowSpacingTopTop = 2.8
      rowSpacingBotBot = 1.4
      abuttableTopTop = 1
      abuttableBotBot = 1
}
```

Cell Row Spacing Rules for Non-Double-Back Cell Rows

When not using double-back cell rows in your floorplan, you specify a row spacing rule between the top edge and the bottom edge, as shown in Figure 1-5.

Figure 1-5 Row Spacing Rule for Non-Double-Back Cells



The attributes used to define these rules are

`rowSpacingTopBot`

The spacing between the top edge and the bottom edge when you are not using double-back cell rows in your floorplan.

`abuttableTopBot`

Whether or not the top edge and the bottom edge can be abutted when you are not using double-back cell rows in your floorplan. Valid values are 1 or 0.

Example 1-20 shows a `PRRule` section for non-double-back cell rows.

Example 1-20 PRRule Section for Non-Double-Back Cell Rows

```
PRRule {
      rowSpacingTopBot = 2.8
      abuttableTopBot = 0
}
```

DensityRule Section

A `DensityRule` section defines the metal density rules used by the `insert_metal_filler` command for a specific metal layer. The technology file should contain a `DensityRule` section for each metal layer.

Use the following attributes to define the metal density rules:

`layer`

The metal layer.

`windowSize`

The size of the window for the density check. By default, the window step size is half of the defined windowSize.

The check setup is assumed to be half of the window size.

`minDensity`

The minimum percentage of metal allowed in the window.

`maxDensity`

The maximum percentage of metal allowed in the window.

`maxGradientDensity`

The maximum percentage difference between the fill density of adjacent windows.

For more information about the metal density rules, see "Metal Density Rules" on page 2-83

Example 1-21 shows a sample `DensityRule` section.

Example 1-21 DensityRule Section

```
DensityRule {
   layer =  "M1"
   windowSize =  200
   minDensity =  20
   maxDensity =  80
}
```

SlotRule Section

A `SlotRule` section defines the slotting rules used by the `slot_wires` command for a specific metal layer. The technology file should contain a `SlotRule` section for each metal layer.

Use the following attributes to define the slotting rules:

`layerNumber`

The layer number.

`lengthThreshold`

The length threshold of metal wires requiring slotting.

`widthThreshold`

The width threshold of metal wires requiring slotting.

`minSlotWidth`

The minimum width of the slot.

`maxSlotWidth`

The maximum width of the slot.

`minSlotLength`

The minimum length of the slot.

`maxSlotLength`

The maximum length of the slot.

`minSideSpace`

The minimum space between adjacent slots in a direction perpendicular to the wire (current flow) direction.

`maxSideSpace`

The maximum space between adjacent slots in a direction perpendicular to the wire (current flow) direction.

`minSideClearance`

The minimum space from the side edge of a wire to its outermost slot.

`maxSideClearance`

The maximum space from the side edge of a wire to its outermost slot.

`minEndSpace`

The minimum space between adjacent slots in the wire (current flow) direction.

`maxEndSpace`

The maximum space between adjacent slots in the wire (current flow) direction.

`minEndClearance`

The minimum space from the end edge of a wire to its outermost slot.

`maxEndClearance`

The maximum space from the end edge of a wire to its outermost slot.

`maxMetalDensity`

The maximum metal density allowed for a slotted wire.

Example 1-22 shows a sample `SlotRule` section.

Example 1-22 SlotRule Section

```
SlotRule "" {
      layerNumber = 1
      lengthThreshold = 0
      widthThreshold = 0
      minSlotWidth = 0
      maxSlotWidth = 0
      minSlotLength = 0
      maxSlotLength = 0
      minSideSpace = 0
      maxSideSpace = 0
      minSideClearance = 0
      maxSideClearance = 0
      minEndSpace = 0
      maxEndSpace = 0
      minEndClearance = 0
      maxEndClearance = 0
      maxMetalDensity = 0
}
```

2

Routing Design Rules

IC Compiler supports routing design rules for advanced technologies extending to 90 nm, 65 nm, 45 nm, and 32 nm. All the design rules are supported by both the detail route operation and the search-and-repair operation, unless otherwise noted.

For the classic router, most of the routing design rules are defined in the technology file. Some of the rules are defined with variables or detail route options or have additional controls that are defined with variables or detail route options that you enter in the command window during an IC Compiler session.

For Zroute, all of the routing design rules must be defined in the technology file.

This chapter contains the following sections:

Minimum Area Rules

The minimum area rules are described in the following sections:

- Minimum Length Rule
- General Minimum Area Rule
- Two-Stage Special Minimum Area Rule

Minimum Length Rule

The minimum length rule specifies the minimum length of a polygon. For a single rectangle, the longest dimension must meet this rule. For a polygon composed of two or more rectangles, the length of the bounding box that encloses the entire geometry must meet this rule.

To define the minimum length rule, use the `minLength` attribute in a metal `Layer` section of the technology file. For example,

```
Layer "M1" {
   minLength = 0.35
}
```

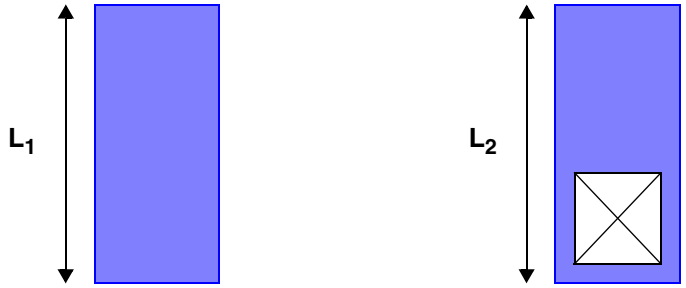
A minimum length violation can be fixed by adding metal stubs or by rerouting the wire to meet the `minLength` value.

By default, the `minLengthMode` attribute is set to 0, and the minimum length rule is enforced for all shapes, irrespective of vias. You can change this behavior by setting the `minLengthMode` attribute to 1, which causes the check to be performed only on segments that fully enclose vias:

```
Technology {
   minLengthMode = 1
}
```

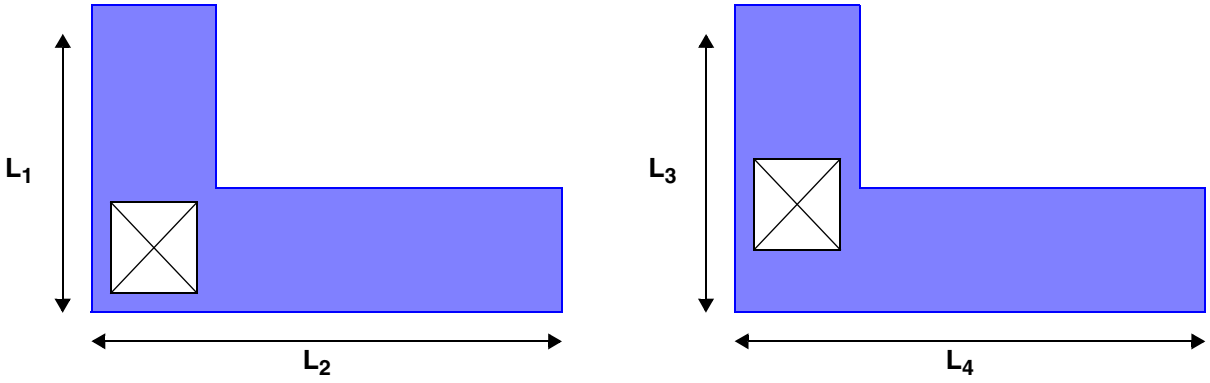
In Figure 2-1, by default, both L_1 and L_2 must be at least the minimum length, L_{min}, which is specified by `minLength`. However, if the `minLengthMode` attribute is set to 1, L_1 is not checked because it does not enclose a via.

Figure 2-1 Minimum Length Rule for Enclosed Vias



With the `minLengthMode` attribute set to 1, if a via is fully enclosed by more than one wire segment in a given layer, the minimum length check is performed only on the longest segment. For example, in Figure 2-2, L_2 is longer than L_1, so the minimum length check is performed only on L_2 and not on L_1. However, L_3 fully encloses the via and L_4 does not, so the minimum length check is performed on L_3 and not on L_4.

Figure 2-2 Minimum Length Rule for Multiple Wire Segments Enclosing a Via



General Minimum Area Rule

The `minArea` attribute defines the minimum area for all geometries. Specify this attribute in a metal `Layer` section of the technology file. For example,

```
Layer "M1" {
   minArea = 0.2
}
```

This general rule works with both Zroute and the classic router.

Two-Stage Special Minimum Area Rule

The two-stage minimum area rule allows you to define a smaller minimum area for shapes in general and a larger minimum area for special shapes. Applying different minimum-area rules for different shapes can help conserve routing resources. Without this capability, you would need to specify a larger minimum area to cover all kinds of shapes.

Use the `minArea` attribute to define a general minimum area for all geometries. Use the `specialMinArea` attribute to define a different, larger minimum area for special shapes. A special shape is a polygon having the following characteristics:

- It is not a rectangle
- All edge lengths are less than a specified threshold, `minAreaEdgeThrehsold`
- It cannot cover a rectangle measuring `minAreaEdgeThreshold` by `minWidth`

All three of these conditions must be met for the `specialMinArea` rule to apply. If a shape is a rectangle, or has any edge longer than `minAreaEdgeThreshold`, or is able to completely cover a rectangle measuring `minAreaEdgeThreshold` by `minWidth`, then the `specialMinArea` rule does not apply; the lower minimum area `minArea` rule is used instead. The `specialMinArea` setting must be greater than the `minArea` setting.

You enable the two-stage minimum area rule by setting the `minAreaMode` attribute to 1 in the `Technology` section of the technology file. By default, it is set to 0 and the router checks only the `minArea` rule. With the attribute set to 1, the router honors the `specialMinArea` value for special shapes and the `minArea` value for ordinary shapes.

In the following example, the two-stage minimum area rule is enabled. For layer M1, the general minimum area is 0.02, but for special shapes, the minimum area is 0.05. A special shape is a polygon that is not rectangular, is composed entirely of segments less than 0.20 in length, and cannot cover a rectangle measuring 0.20 by 0.07.

```
Technology {
   minAreaMode = 1
   }

Layer "M1"  {
   minWidth             = 0.07
   minArea              = 0.02
   minAreaEdgeThreshold = 0.20
   specialMinArea       = 0.05
   ...
   }
```

This rule is supported by both Zroute and the classic router. It is not necessary to specify `minAreaMode = 1` for Zroute, but it is required for the classic router.

Minimum Enclosed Area Rule

The minimum enclosed area rule defines the minimum area enclosed by ring-shaped wires or vias. You define this rule in a `Layer` section of the technology file by specifying the `minEnclosedArea` attribute for the associated layer. Define this attribute in a metal `Layer` section of the technology file. For example,

```
Layer "M1" {
   minEnclosedArea       = 0.2
}
```

IC Compiler honors the minimum enclosed area rule by avoiding the creation of fat wires. Use the `fatTblThreshold` attribute to specify the thresholds that the router uses to determine whether a metal wire is fat. Use the `fatTblMinEnclosedArea` attribute to specify the minimum enclosed area for thin metal; the router uses the first nonzero value in the list. These attributes are defined in a `Layer` section of the technology file.

In the following example, the router avoids creating any metal wires with widths greater than or equal to 0.155.

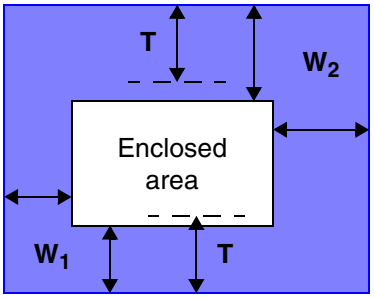
```
Layer "M1" {
   fatTblDimension     = 3
   fatTblThreshold     = (0, 0.155, 1.605)
   fatTblMinEnclosedArea = (0.3, 1.0, 1.0)
}
```

You can also set the `fatTblMinEnclosedAreaMode` attribute in the `Technology` section of the technology file. This attribute determines the mode in which the router checks for and avoids a violation:

- When `fatTblMinEnclosedAreaMode = 0`, which is the default, the fat metal minimum enclosed area mode is triggered when any of the surrounding metal is narrower than the threshold width.
- When `fatTblMinEnclosedAreaMode = 1`, the fat metal minimum enclosed area mode is triggered only when all of the surrounding metal is less than the threshold width.

In Figure 2-3, the width of the metal surrounding the enclosed area is W1 on two sides and W2 on the other two sides. If W1 is less than the fat threshold T and W2 is more than the fat threshold T, the `fatTblMinEnclosedAreaMode` setting determines whether the minimum enclosed area rule is applied. If the mode is set to 0 (the default), the rule is applied because there is at least one surrounding width below the threshold. If the mode is set to 1, the rule is not applied because not all of the surrounding widths are below the threshold.

Figure 2-3 Minimum Enclosed Area Rule Modes



This rule can be useful when standard cells have ring-shaped pins. The cells themselves satisfy the rule, possibly using a minimum-sized enclosed area. When a router connects to these pins, it must make the connection in such a manner that no fat wire is created on any edges or on all edges of the enclosed area, depending on the technology requirements, so that there is no requirement to have a larger enclosed area.

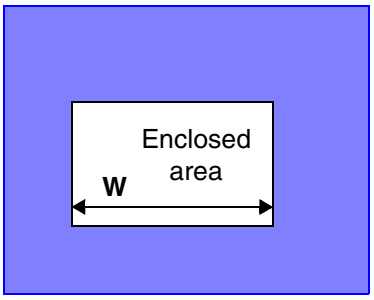
Minimum Enclosed Width Rule

The `minEnclosedWidth` attribute specifies the minimum width of an enclosed area for the given routing layer. Define this attribute in a metal `Layer` section of the technology file. For example,

```
Layer "M1" {
    minEnclosedWidth = 0.7
}
```

The width of an enclosed area checked by this rule is the larger of the two dimensions of the area, as shown in Figure 2-4.

Figure 2-4 Minimum Enclosed Width



This rule can be useful when standard cells have ring-shaped or U-shaped pins. The cells themselves satisfy the rule. When a router connects to these pins, it must make the connection in such a manner that no enclosed area is created, or if an enclosed area is created or changed, the larger dimension of the enclosed area is at least as large as the specified minimum width.

Minimum Edge Rules

The minimum edge rules are described in the following sections:

- Minimum Edge Mode
- Maximum Total Number of Short Edges Rule
- Maximum Total Short Edge Length Rule
- Special Notch Rule
- Three-Adjacent-Edge Minimum Length Rule
- Short Edge to End-of-Line Rule

Minimum Edge Mode

You control the scope of the check for the minimum edge rules by setting the `minEdgeMode` attribute in the `Technology` section of the technology file. For example,

```
Technology {
   minEdgeMode = 0
}
```

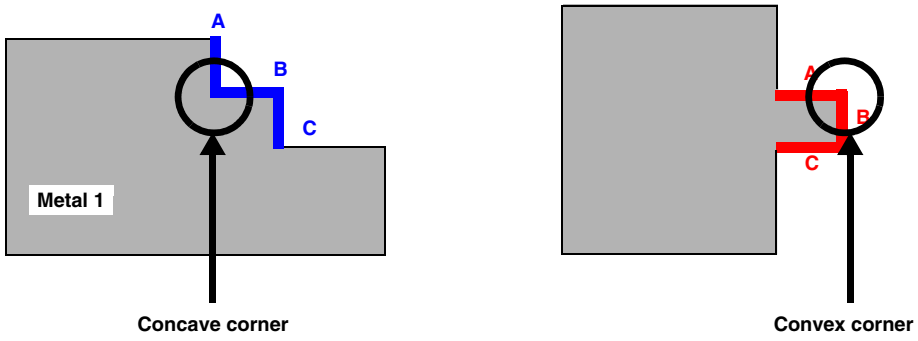
When the mode is set to 0 (the default), a small concave corner must be present to trigger a violation. A small concave corner is formed by two adjacent edges that are both less than the minimum edge length. When the mode is set to 1, a small concave corner need not be present to trigger a violation; a convex corner alone can trigger a violation.

Maximum Total Number of Short Edges Rule

The maximum total number of short edges rule specifies the maximum number of consecutive short edges. The scope of the rule is controlled by the `minEdgeMode` attribute in the `Technology` section of the technology file. When `minEdgeMode` is 0, violations are triggered only when the short edges include a concave corner. When `minEdgeMode` is 1, a concave corner is not necessary to trigger a violation; a convex corner alone can trigger a violation. For example, in Figure 2-5, assume edges A, B, and C are all short edges and the

maximum number of consecutive short edges is 2. This rule is violated in the left-hand figure, regardless of the `minEdgeMode` setting, because it contains a concave corner, but the rule is violated in the right-hand figure only when `minEdgeMode` is 1.

Figure 2-5 Minimum Edge Examples



To define this rule, set the following attributes in a metal `Layer` section of the technology file:

- `minEdgeLength`

  This attribute specifies the maximum length that defines a short edge.

- `maxNumMinEdge`

  This attribute specifies the number of consecutive edges to check. This rule is violated only when the number of consecutive short edges is greater than `maxNumEdge`.

For example, to define a short edge as an edge less than 0.07 microns and the maximum number of consecutive short edges as 2, enter

```
Layer "M1" {
    minEdgeLength = 0.07
    maxNumMinEdge = 2
}
```

Maximum Total Short Edge Length Rule

The maximum total minimum edge length rule specifies the maximum total length of consecutive short edges. The scope of the rule is controlled by the `minEdgeMode` attribute in the `Technology` section of the technology file. When `minEdgeMode` is 0, violations are triggered only when the short edges include a concave corner. When `minEdgeMode` is 1, a concave corner is not necessary to trigger a violation; a convex corner alone can trigger a violation. For example, in Figure 2-5 on page 2-9, assume edges A, B, and C are all short edges and the total length of these edges exceeds the maximum total short edge length. This rule is violated in the left-hand figure regardless of the `minEdgeMode` setting because it contains a concave corner, but the rule is violated in the right-hand figure only when `minEdgeMode` is 1.

To define this rule, set the following attributes in a metal `Layer` section of the technology file:

- `minEdgeLength`

  This attribute specifies the maximum length that defines a short edge.

- `maxTotalMinEdgeLength`

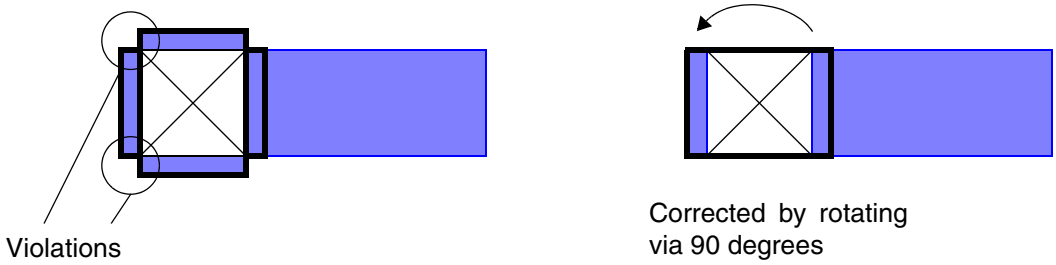
  This attribute specifies the maximum total length of the consecutive short edges.

For example, to define a short edge as an edge less than 0.07 microns and the maximum total length of consecutive short edges as 0.11 microns, enter

```
Layer   "M1" {
    minEdgeLength = 0.07
    maxTotalMinEdgeLength  = 0.11
}
```

Figure 2-6 shows an example of an application of the rule. On the left, two metal enclosures of stacked vias are perpendicular to each other, causing the short edge length errors circled in the diagram. This can be corrected by rotating one of the vias by 90 degrees, as shown on the right, causing the vias to coincide perfectly.

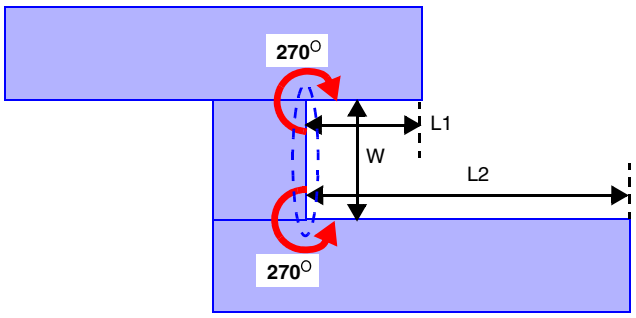
Figure 2-6 Total Short Edge Length Rule



Special Notch Rule

The special notch rule specifies the minimum width (W in Figure 2-7) of a notch when at least one of the adjacent edges (L1 and L2 in Figure 2-7) is less than a specified length.

Figure 2-7 Special Notch Rule Example



To define this rule, set the following attributes in a metal `Layer` section of the technology file:

- `minEdgeLength2`

  This attribute specifies the length threshold for applying this rule. This rule applies if at least one of the edges adjacent to the notch is less than `minEdgeLength2`.

- `minEdgeLength3`

  This attribute specifies the minimum width of the notch, W.

For example, to specify that the minimum notch width is 0.26 microns when at least one of the adjacent edges is less than 0.4 microns, enter the following:

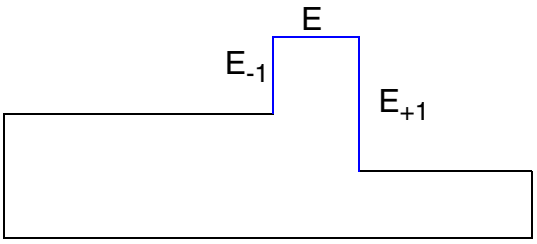
```
Layer   "M1"  {
   minEdgeLength2 = 0.4
   minEdgeLength3 = 0.26
}
```

Special notch rule violations can be fixed by rerouting wires, rotating vias, or adding stubs.

Three-Adjacent-Edge Minimum Length Rule

The three-adjacent-edge minimum length rule specifies the minimum lengths of the edges E_{-1} and E_{+1} connected to a short edge E, as shown in Figure 2-8.

Figure 2-8 Three Adjacent Edges Considered



To define this rule, set the following detail route options:

- `minEdgeLengthMode`

  This option must be set to 0 for this rule.

- `MxMinEdgeLength4`

  This option specifies the maximum length for edge E to be considered a short edge.

- `MxMinEdgeLength5`

  This option specifies the minimum length of edge E_{-1}. This value must be less than or equal to the value specified for `MxMinEdgeLength6`.

- `MxMinEdgeLength6`

  This option specifies the minimum length of edge E_{+1}.

where *x* is the metal layer to which the rule applies. Valid values for *x* are 1 through 15. The settings of these detail route options are saved with the cell.

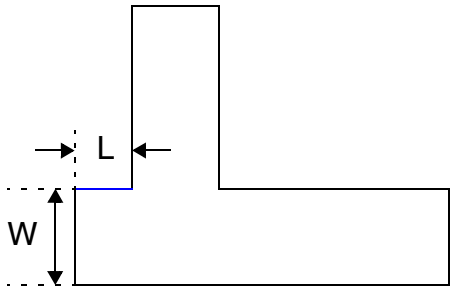
For example, to define the rule for the metal 1 layer such that when E is less than 0.4 microns, E_{-1} must be at least 0.26 microns and E_{+1} must be at least 0.3 microns, enter the following commands:

```
icc_shell> set_droute_options -name minEdgeLengthMode -value 0
icc_shell> set_droute_options -name M1MinEdgeLength4 -value 0.4
icc_shell> set_droute_options -name M1MinEdgeLength5 -value 0.26
icc_shell> set_droute_options -name M1MinEdgeLength6 -value 0.3
```

Short Edge to End-of-Line Rule

The short edge to end-of-line rule specifies the minimum length of an edge adjacent to a line-end edge that is less the specified length. In Figure 2-9, when the width of the line-end is less than W, the short edge L must be at least a specified value.

Figure 2-9 Short Edge to End-of-Line Rule



You define this rule in a `Layer` section by using the `minEdgeLengthTblSize` and `minEdgeLengthTbl` attributes.

For the classic router, you can also define this rule by using the `minEdgeLengthMode`, `Mn`MinEdgeLength4, and `Mn`MinEdgeLength5 detail route options.

When the `minEdgeLengthMode` detail route option is set to 1,

- `Mn`MinEdgeLength4 defines the short edge length
- `Mn`MinEdgeLength5 defines the end-of-line width threshold that triggers the rule

For example, to set the threshold W to 0.4 and the minimum short edge to 0.26 for layer M4, you would use the following commands:

```
icc_shell> set_droute_options -name minEdgeLengthMode -value 1
icc_shell> set_droute_options -name M4MinEdgeLength4 -value 0.26
icc_shell> set_droute_options -name M4MinEdgeLength5 -value 0.4
```

Minimum Spacing Rules

The minimum spacing rules are described in the following sections:

- Minimum Spacing Rule
- U-Shape Spacing Rule
- Via Corner Spacing Rule
- Via Same Net Minimum Spacing Rule
- General Cut Spacing Rule

Minimum Spacing Rule

Use the `minSpacing` attribute in a `Layer` section of the technology file to specify the minimum spacing allowed for that layer between the edges of two objects on different nets. This is the default minimum spacing requirement for that layer.

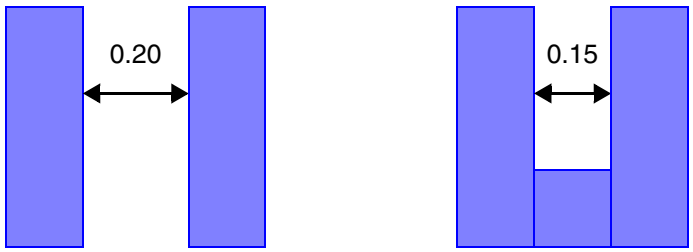
You can specify a smaller minimum spacing rule for objects that are on the same net. Use the `sameNetMinSpacing` attribute in a `Layer` section of the technology file to specify the minimum spacing allowed between the edges of two objects on the same net.

For example,

```
Layer   "M1" {
    minSpacing          = 0.20
    sameNetMinSpacing   = 0.15
}
```

In Figure 2-10, the unconnected wires must have a spacing of at least 0.20, whereas the connected wires must have a spacing of at least 0.15.

Figure 2-10 Unconnected and Connected Minimum Spacing Rule

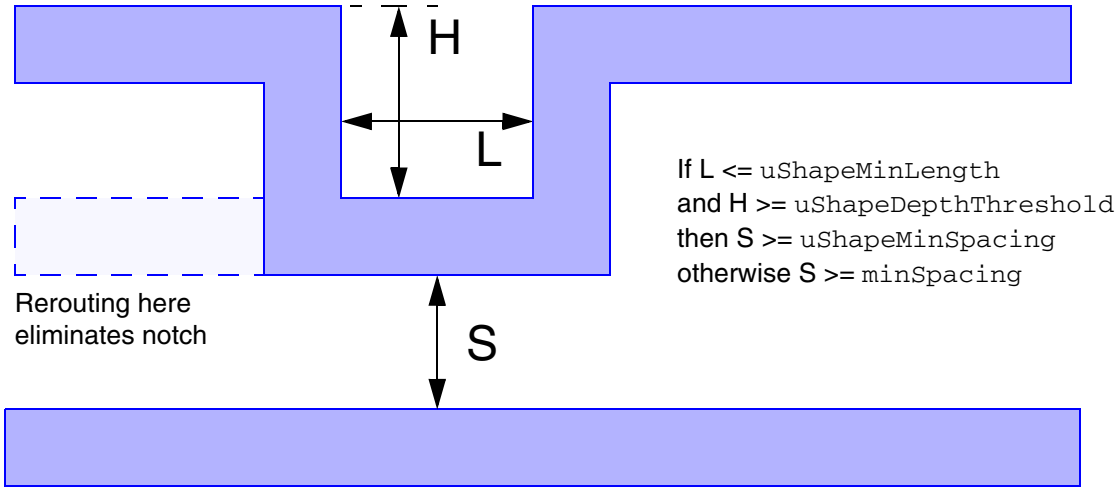


U-Shape Spacing Rule

To increase the minimum spacing between a short U-shaped notch and a neighboring wire, use the `uShapeDepthThreshold`, `uShapeMinLength`, and `uShapeMinSpacing` attributes. Define these attributes in a metal `Layer` section of the technology file.

In Figure 2-11, the rule defines the minimum spacing (S), as specified with `uShapeMinSpacing`, between two edges when the height (H) is greater than or equal to `uShapeDepthThreshold` and the length (L) is less than `uShapeMinLength`. Otherwise, the smaller default minimum spacing specified with `minSpacing` applies.

Figure 2-11 U-Shape Spacing Rule



For example,

```
Layer   "M1" {
   uShapeMinSpacing     = 0.13
   uShapeDepthThreshold = 0.05
   uShapeMinLength       = 0.16
}
```

Via Corner Spacing Rule

Use the `cornerMinSpacing` attribute to specify the minimum corner-to-corner spacing allowed between two vias, which must be smaller than the minimum spacing value specified with the `minSpacing` attribute.

For corner spacing on the same via layer, define this attribute in a via `Layer` section of the technology file. For example,

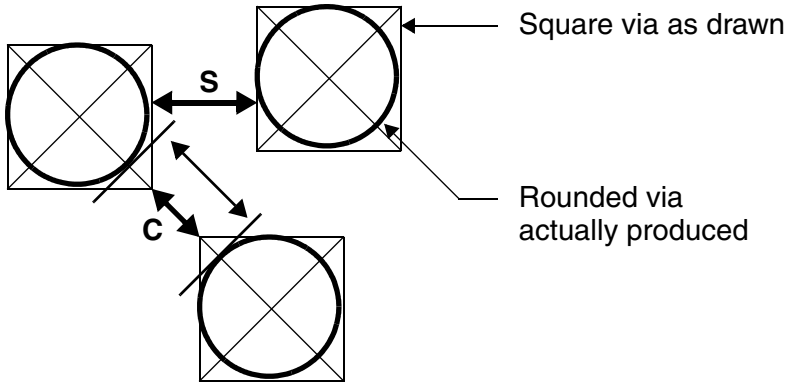
```
Layer "VIA1" {
    minSpacing       = 0.20
    cornerMinSpacing = 0.12
}
```

For corner spacing between different via layers, define this attribute in the `DesignRule` section of the technology file. For example,

```
DesignRule {
   layer1              = "VIA1"
   layer2              = "VIA2"
   minSpacing          = 0.20
   cornerMinSpacing    = 0.12
}
```

The reason for allowing closer spacing between corners is that the edges of real physical vias are typically rounded within the boundaries of the drawn vias, as depicted in Figure 2-12. Therefore, the minimum spacing requirement C between drawn corners might be less than the minimum spacing requirement S between side-by-side edges. By allowing a closer spacing C between drawn corners, routing resources can be conserved.

Figure 2-12 Drawn Vias Versus Actual Physical Vias

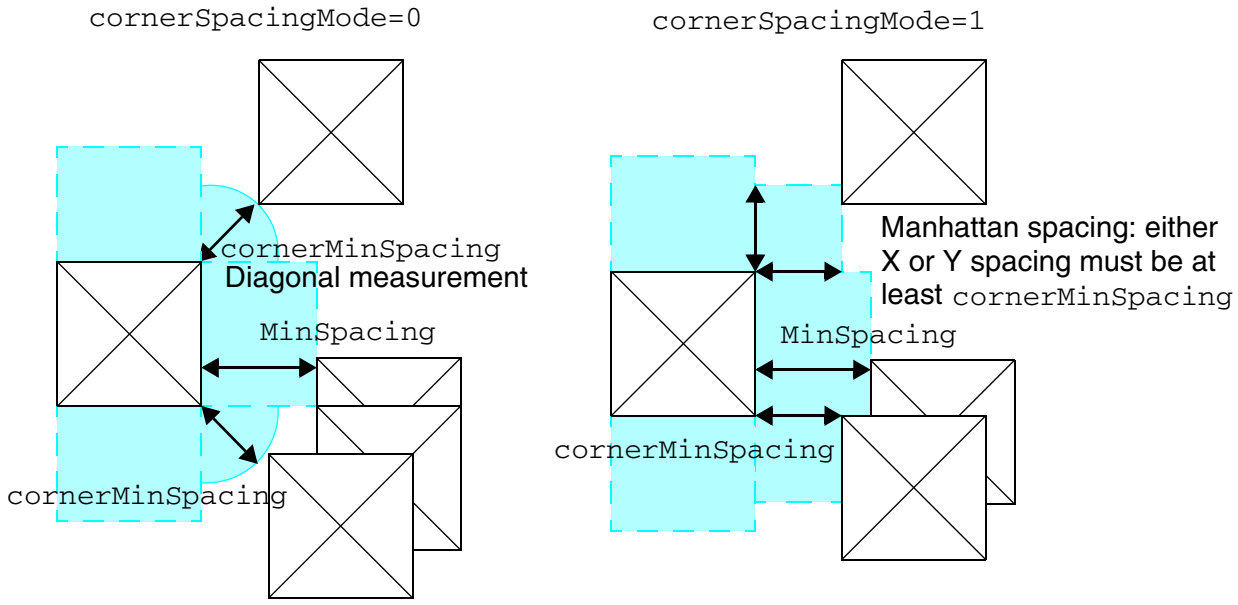


You can control the measurement method, either diagonal or Manhattan, for corner-to-corner spacing between vias by setting the `cornerSpacingMode` attribute in the `Technology` section of the technology file. By default (`cornerSpacingMode` = 0), IC Compiler measures the diagonal distance. To use Manhattan distance instead, set the `cornerSpacingMode` attribute to 1. For example,

```
Technology {
   cornerSpacingMode = 1
}
```

Figure 2-13 demonstrates the difference between diagonal and Manhattan distance measurements, as determined by the `cornerSpacingMode` setting. For the case where the outside edges of two vias line up exactly, the `minSpace` value is used if `cornerSpacingMode` is set to 0, or the `cornerMinSpacing` value is used if `cornerSpacingMode` is set to 1.

Figure 2-13 Diagonal and Manhattan Distance Measurements



Via Same Net Minimum Spacing Rule

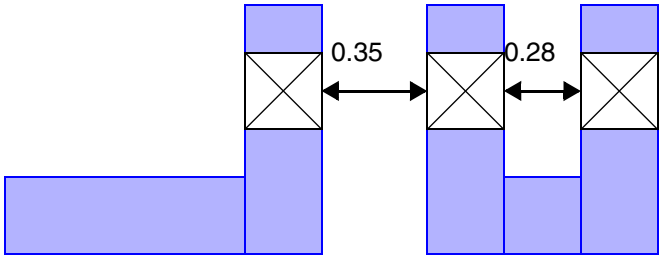
Use the `sameNetMinSpacing` attribute to override the `minSpacing` default value and set a smaller value for the minimum spacing between two vias belonging to the same net.

Define this attribute in a via `Layer` section of the technology file. For example,

```
Layer "VIA5"  {
    sameNetMinSpacing  = 0.28
    minSpacing         = 0.35
}
```

In Figure 2-14, the minimum spacing between the two vias belonging to different nets is 0.35, whereas the minimum spacing between the two vias belonging to the same net is 0.28.

Figure 2-14 Unconnected and Connected Via Minimum Spacing Rule



General Cut Spacing Rule

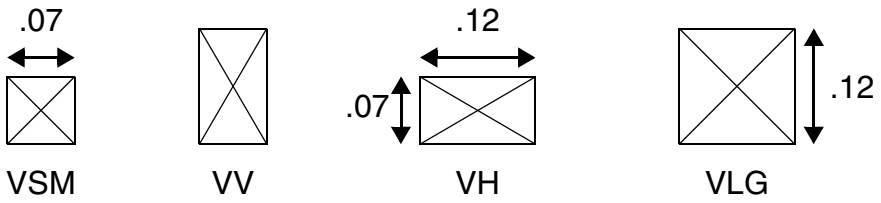
You can specify the minimum spacing between cuts (vias and contacts) based on the cut dimensions and the net and wire-segment relationships of the cuts. To use this rule, you specify the dimensions of the cuts and assign a name to each cut size. Then you create a table of minimum spacing values that apply to each possible combination of named cut sizes. You can specify separate minimum spacing values for cuts in the same net, in different nets, in the same metal segment, and in different metal segments.

Note:

This rule is supported only by Zroute. For a similar rule supported by the classic router, see the previous sections, "Via Corner Spacing Rule" on page 2-15 and "Via Same Net Minimum Spacing Rule" on page 2-17.

Usage of this rule is best explained by example. Suppose that you use the four cut sizes shown in Figure 2-15 in layer Via1.

Figure 2-15 Specifying Cut Sizes for General Cut Spacing Rule



You assign the names to the four via sizes for layer Via1 using the following syntax:

```
Layer "Via1" {
  cutTblSize = 4
  cutNameTbl =    (VSM, VV, VH, VLG)
  cutWidthTbl =   (.07, .07, .12, .12)
  cutHeightTbl = (.07, .12, .07, .12)
  }
```

Suppose that you use the same cut sizes in layer Via2. You similarly assign the same set of names to the same four via sizes in layer Via2 as follows:

```
Layer "Via2" {
  cutTblSize = 4
  cutNameTbl =    (VSM, VV, VH, VLG)
  cutWidthTbl =   (.07, .07, .12, .12)
  cutHeightTbl = (.07, .12, .07, .12)
  }
```

Now you need to specify the spacing between just the square vias in layers Via1 and Via2, with different spacing values for vias in the same net versus vias in different nets. To specify these spacing values, you use the following syntax:

```
DesignRule {
  layer1 = "Via1"
  layer2 = "Via2"
  cut1TblSize = 2
  cut2TblSize = 2
  cut1NameTbl = (VSM, VLG)
  cut2NameTbl = (VSM, VLG)
  orthoSpacingExcludeCornerTbl = (0, 1,
                                  1, 0)
  sameNetXMinSpacingTbl= (0.10, 0.10,
                          0.10, 0.12)
  diffNetXMinSpacingTbl= (0.11, 0.11,
                          0.11, 0.13)
  }
```

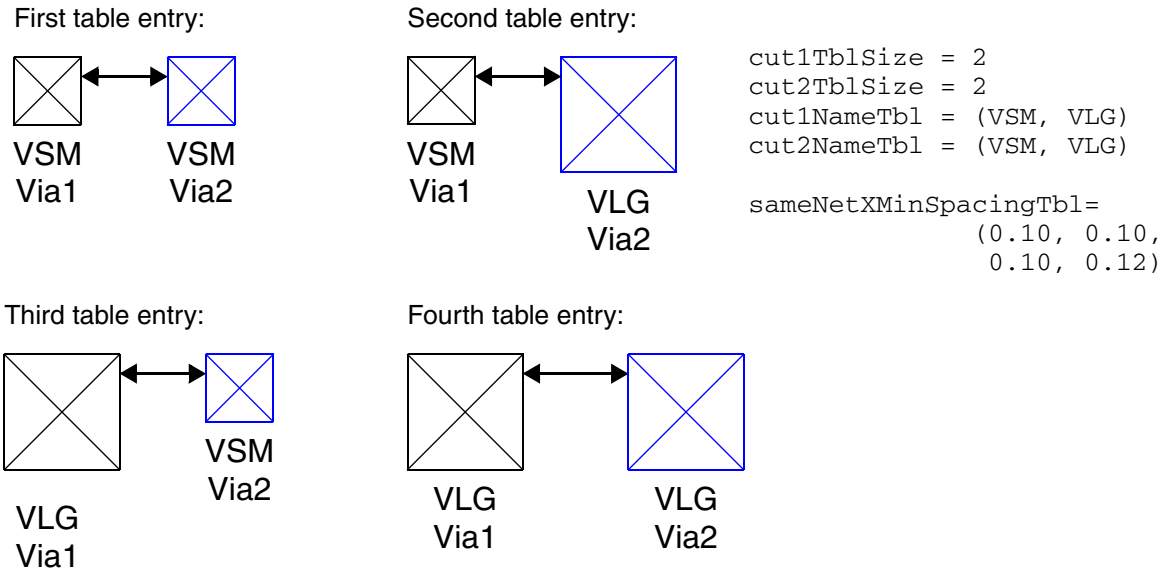
The `layer1` and `layer2` attributes specify the cut layers involved in the rule. The two layers can be different layers or the same layer. The rule specifies the minimum distance between cuts on the two specified layers or between cuts in the same layer. The `cut1TblSize` and `cut2TblSize` attributes specify the number of named cuts in each layer for which the current rule applies. The `cut1NameTbl` and `cut2NameTbl` attributes list the named cuts for which the current rule applies. The total number of entries in each table is the product of the `cut1TblSize` and `cut2TblSize` attributes, which is equal to the total number of combinations between the named cuts in the two lists.

In this example, the rule specifies the minimum spacing between cuts in layer Via1 to cuts in layer Via2. Each table has four entries, each corresponding to a combination of named cuts taken in order from the `cut1NameTbl` and `cut2NameTbl` tables:

- VSM in layer Via1 to VSM in layer Via2
- VSM in layer Via1 to VLG in layer Via2
- VLG in layer Via1 to VSM in layer Via2
- VLG in layer Via1 to VLG in layer Via2

The `orthoSpacingExcludeCornerTbl` attribute is a table of Boolean values that specifies whether the rule is extended beyond the corners of the cut. The `sameNetXMinSpacingTbl` and `diffNetXMinSpacingTbl` attributes are tables that specify the minimum spacing values between the named cuts for same-net and different-net cuts, respectively. For an example, see Figure 2-16.

Figure 2-16 Cut-to-Cut Spacing Specified in a 2-by-2 Table



The following table-based attributes can be used to specify the edge-to-edge X spacing, edge-to-edge Y spacing, corner-to-corner spacing, or center-to-center spacing of two cuts in the same net or in different nets, respectively:

```
sameNetXMinSpacingTbl
sameNetYMinSpacingTbl
sameNetCornerMinSpacingTbl
sameNetCenterMinSpacingTbl

diffNetXMinSpacingTbl
diffNetYMinSpacingTbl
diffNetCornerMinSpacingTbl
diffNetCenterMinSpacingTbl
```

Similarly, the following table-based attributes can be used to specify the edge-to-edge X spacing, edge-to-edge Y spacing, corner-to-corner spacing, or center-to center spacing of two cuts in the same wire segment or in different wire segments, respectively:

```
sameSegXMinSpacingTbl
sameSegYMinSpacingTbl
sameSegCornerMinSpacingTbl
sameSegCenterMinSpacingTbl

diffSegXMinSpacingTbl
diffSegYMinSpacingTbl
diffSegCornerMinSpacingTbl
diffSegCenterMinSpacingTbl
```

If you do not want to specify a minimum spacing value for a particular combination of named cuts, enter –1.0 in the corresponding position in the minimum spacing table. Entering a negative value prevents the rule from checking that particular combination.

You can define different rules in separate DesignRule sections as long as the sections have different cut name tables. For example, you can define cut rules `sameNetXMinSpacing` and `diffNetXMinSpacing` for all cuts in one table:

```
DesignRule {
  layer1 = "VIA1"
  layer2 = "VIA2"
  cut1TblSize = 4
  cut2TblSize = 4
  cut1NameTbl = (VSM, VV, VH, VLG)
  cut2NameTbl = (VSM, VV, VH, VLG)
  sameNetXMinSpacingTbl = (0.10,0.10,0.12,0.10,
                           0.10,0.10,0.15,0.10,
                           0.12,0.15,0.15,0.12,
                           0.10,0.10,0.12,0.12)
  diffNetXMinSpacingTbl = (0.10,0.10,0.12,0.10,
                           0.10,0.10,0.15,0.10,
                           0.12,0.15,0.15,0.12,
                           0.10,0.10,0.12,0.12)
  }
```

At the same time, you can define a different cut rule based on wire segment relationships using `diffSegCenterMinSpacing` in another DesignRule section. The table size can be different, as in the following example:

```
DesignRule {
  layer1 = "VIA1"
  layer2 = "VIA2"
  cut1TblSize = 2
  cut2TblSize = 2
  cut1NameTbl = (VSM, VLG)
  cut2NameTbl = (VSM, VLG)
  diffSegCenterMinSpacing  = (0.09, 0.09,
                              0.09, 0.11)
  }
```

You can define cut rules for cuts on the same layer, as in the following example:

```
DesignRule {
  layer1 = "VIA2"
  layer2 = "VIA2"
  cut1TblSize = 2
  cut2TblSize = 2
  cut1NameTbl = (VSM, VLG)
  cut2NameTbl = (VSM, VLG)
  diffSegCenterMinSpacing = (0.10, 0.10,
                             0.10, 0.12)
```

```
}
```

The cut rule minimum spacing values should meet the following requirements when the two cuts are on the same layer:

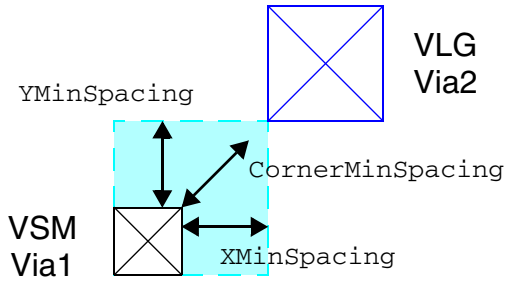
```
sameNetXMinSpacingTbl[i] >= xMinSpacing >= minSpacing
sameNetYMinSpacingTbl[i] >= yMinSpacing >= minSpacing
diffNetXMinSpacingTbl[i] >= xMinSpacing >= minSpacing
diffNetYMinSpacingTbl[i] >= yMinSpacing >= minSpacing
sameSegXMinSpacingTbl[i] >= max(minCutSpacing, xMinSpacing)
sameSegYMinSpacingTbl[i] >= max(minCutSpacing, yMinSpacing)
diffSegXMinSpacingTbl[i] >= xMinSpacing >= minSpacing
diffSegYMinSpacingTbl[i] >= yMinSpacing >= minSpacing
```

where i = 0, 1, ..., table_size – 1

table_size = (`cut1TblSize`) x (`cut2TblSize`)

At a corner area between two cuts, if both `XMinSpacing`/`YMinSpacing` and `CornerMinSpacing` are defined, by default, the router checks both types of rules, as shown in Figure 2-17.

Figure 2-17 XMinSpacing and YMinSpacing Enforced at Corners



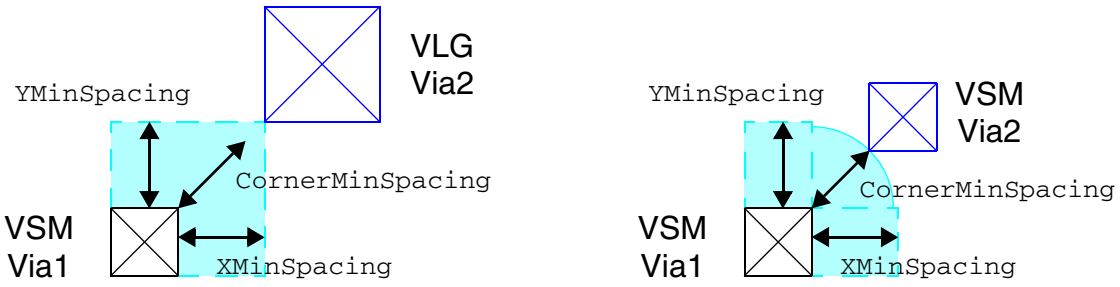
To suppress checking of `XMinSpacing`/`YMinSpacing` at corners and allow only `CornerMinSpacing` to be checked, set the `orthoSpacingExcludeCornerTbl` attribute to 1 for the corresponding check.

In the following example, the router checks both `XMinSpacing`/`YMinSpacing` and `CornerMinSpacing` between cuts VSM and VLG in corner areas for cuts belonging to the same net, whereas it checks only `CornerMinSpacing` between two VSM cuts or between two VLG cuts belonging to the same net. This rule checking is shown in Figure 2-18.

```
DesignRule {
  layer1 = "VIA1"
  layer2 = "VIA2"
```

```
cut1TblSize = 2
cut2TblSize = 2
cut1NameTbl  =(VSM, VLG)
cut2NameTbl = (VSM, VLG)
orthoSpacingExcludeCornerTbl = (1, 0,
                                0, 1)
sameNetXMinSpacingTbl= (0.10, 0.10
                        0.10, 0.12)
sameNetYMinSpacingTbl= (0.10, 0.10
                        0.10, 0.12)
sameNetCornerMinSpacingTbl= (0.14, 0.14)
                            (0.14, 0.17)
...
}
```

Figure 2-18 XMinSpacing and YMinSpacing Suppressed at Corners



X and Y spacing check enforced at corners

X and Y spacing check suppressed at corners

Fat Metal Spacing Rules

The fat metal spacing rules are described in the following sections:

- Fat Metal Spacing Rule Table
- Fat Metal Orthogonal Spacing Rule
- Fat Metal Parallel Length
- Fat Metal Extension Spacing Rule
- Neighboring Layer Fat Metal Extension Spacing Rule

Fat Metal Spacing Rule Table

The fat metal spacing rule defines the minimum space between two parallel metal segments, based on the width of the metal segments and the parallel length between them.

To define this rule, specify the following attributes in a metal `Layer` section of the technology file:

- `fatTblDimension`

  Specifies the size of the two-dimensional fat table.

- `fatTblThreshold`

  Specifies the thresholds used to determine which rules apply, based on the width of the metal segment. The width is the minimum of the x- or y-length of the metal segment.

  You must specify *n* values, where *n* is the table dimension.

- `fatTblParallelLength`

  Specifies the thresholds used to determine which rules apply, based on the length that two metal segments are parallel to each other.

  You must specify *n* values, where *n* is the table dimension.

- `fatTblSpacing`

  Specifies the minimum spacing requirements for each combination of metal widths and parallel lengths.

  You must specify an *n*x*n* table, where *n* is the table dimension.

For example, assume that your fat metal spacing rule requires the following:

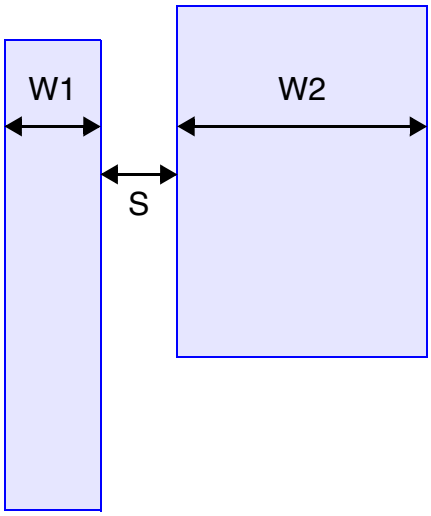
- A minimum spacing of 0.2 microns between any two parallel metal segments.
- A minimum spacing of 0.4 microns between two parallel metal segments when either one is greater than 0.5 microns and the parallel length is greater than 1.5 microns.
- A minimum spacing of 0.8 microns between two parallel metal segments when either one is greater than 2.5 microns and the parallel length is greater than 2.5 microns.

To specify this in the technology file, enter

```
Layer "M1" {
   fatTblDimension      = 3
   fatTblThreshold      = (0.0, 0.501, 2.501)
   fatTblParallelLength = (0.0, 1.501, 2.501)
   fatTblSpacing        = (0.2, 0.4, 0.8,
                           0.4, 0.4, 0.8,
                           0.8, 0.8, 0.8)
}
```

The values in the *n* x *n* table represent the minimum spacing values for different combinations of two fat metal widths. The width ranges are established by the `fatTblThreshold` values. In the foregoing example, the table entries represent the minimum spacing between metal wires as indicated in Figure 2-19.

Figure 2-19 Fat Metal Minimum Spacing Table



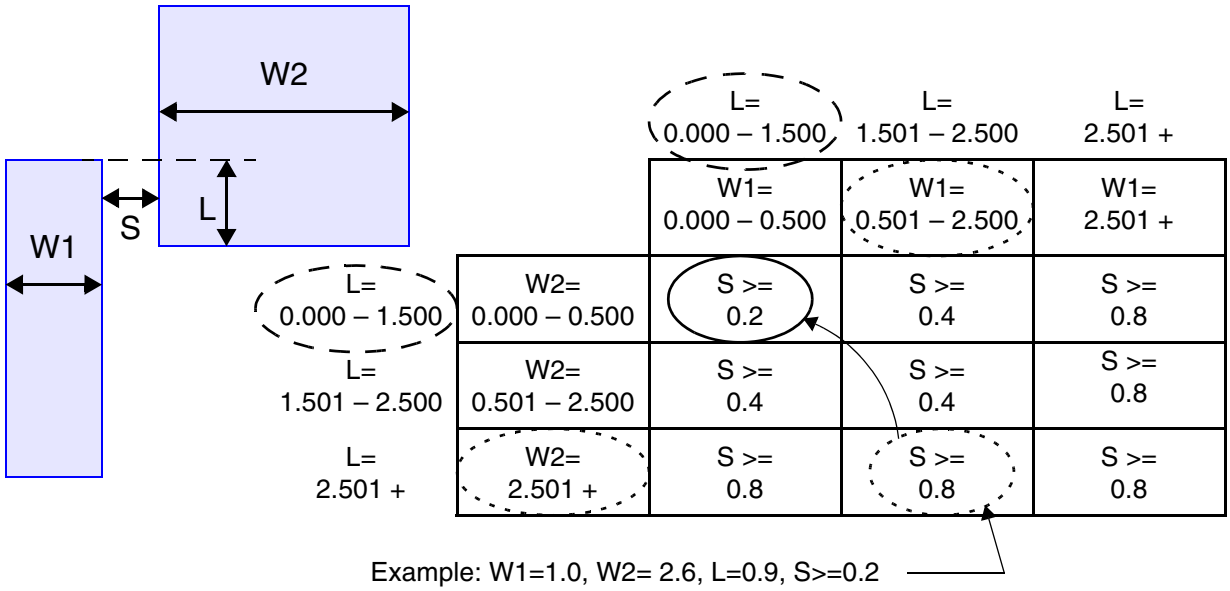
Width	W1= 0.000 – 0.500	W1= 0.501 – 2.500	W1= 2.501 +
W2= 0.000 – 0.500	S >= 0.2	S >= 0.4	S >= 0.8
W2= 0.501 – 2.500	S >= 0.4	S >= 0.4	S >= 0.8
W2= 2.501 +	S >= 0.8	S >= 0.8	S >= 0.8

Example: W1=1.0, W2= 2.6, S>=0.8

The width values should be specified in the distance measurement precision of the technology. For example, if the `Technology` section specifies the `unitLengthName` as `"micron"` and `lengthPrecision` as `1000`, then the thresholds should be written with a precision of .001 micron to specify contiguous ranges, as in the foregoing example.

If the parallel overlap length of the two metal wires is small enough, the minimum spacing requirement is reduced or downgraded as specified by `fatTblParallelLength`. See Figure 2-20 for an example. Because the parallel overlap length L is small, the minimum spacing requirement is downgraded to the designated lesser entry in the table.

Figure 2-20 Parallel Overlap Length Downgrade of Minimum Spacing



You can optionally restrict the amount of the downgrade to no more than one row and one column in the table by setting `fatTblParallelLength` to 1. In that case, in Figure 2-20, the downgrade would be to a minimum spacing of 0.4 rather than 0.2. By default, `fatTblParallelLength` is 0 and there is no restriction on the amount of downgrade that can result from having a smaller parallel overlap.

Fat Metal Orthogonal Spacing Rule

The fat metal orthogonal spacing rule extends the fat metal spacing rule described in the previous section to handle the case of different spacing values in the X and Y directions and to specify the application of the rule at metal corners. The following example demonstrates the usage of this rule.

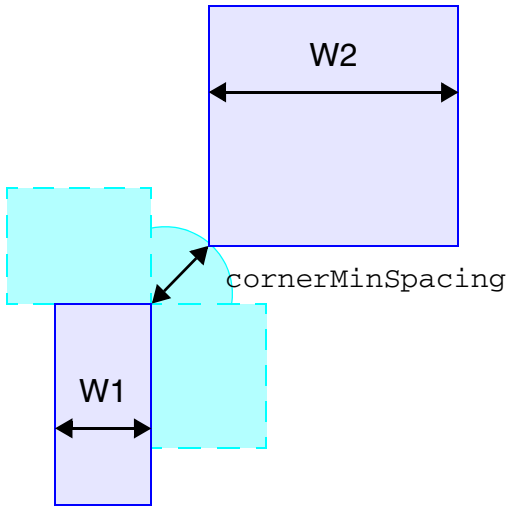
```
Layer "M1" {
  fatTblXDimension = 3
  fatTblYDimension = 3
  fatTblXThreshold = (0.0, 0.501, 2.501)
  fatTblYThreshold = (0.0, 1.001, 5.001)
  fatTblXParallelLength = (0.0, 1.501, 2.501)
  fatTblYParallelLength = (0.0, 1.501, 2.501)
  fatTblXMinSpacing = (0.2, 0.4, 0.8,
                       0.4, 0.4, 0.8,
                       0.8, 0.8, 0.8)
```

```
fatTblYMinSpacing = (0.3, 0.6, 1.2,
                     0.6, 0.6, 1.2,
                     1.2, 1.2, 1.2)
orthoSpacingExcludeCorner = 1
}
```

The spacing attributes are specified separately for the X and Y directions, using different values and possibly different table sizes.

The `orthoSpacingExcludeCorner` attribute specifies whether to extend the spacing check to the corner area for the given layer. The default setting is 0, which enables extension of the spacing check to the corner areas, whereas 1 disables extension of the spacing check to the corner areas. The `cornerMinSpacing` attribute determines the minimum spacing requirement, as illustrated in Figure 2-21. A single, not table-based, corner spacing value applies to the layer.

Figure 2-21 Fat Metal Corner Spacing Check



You can use either the simpler fat table syntax described in the previous section or the orthogonal X-Y syntax just described, but not both within the same technology file.

Note:

If you use the fat metal orthogonal spacing rule and the fat metal extension spacing rule ("Fat Metal Extension Spacing Rule" on page 2-29) in the same technology file, replace the fat metal extension spacing attribute `fatTblExtensionRange` with the two attributes `fatTblXExtensionRange` and `fatTblYExtensionRange`, to ensure correct operation of both rules.

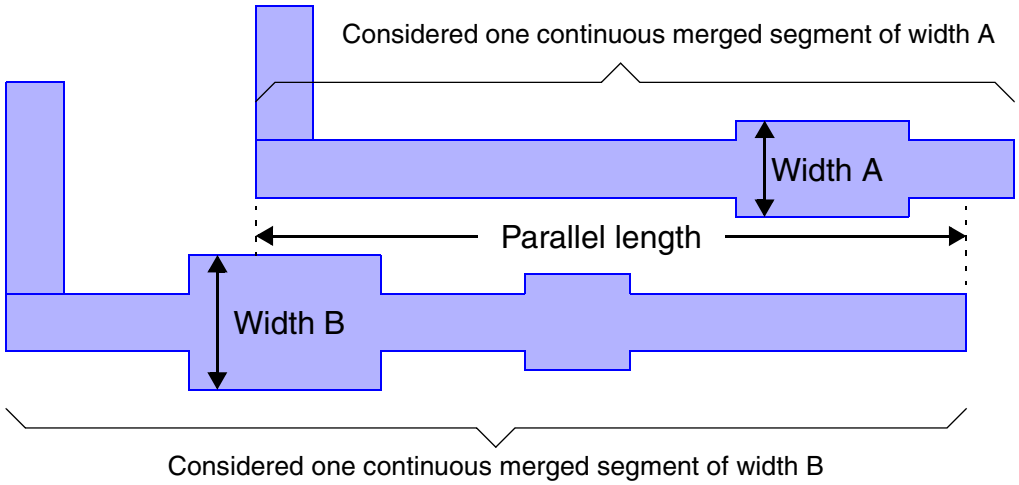
Note:

If you use the fat metal orthogonal spacing rule and the fat metal enclosed minimum area rule (“Minimum Enclosed Area Rule” on page 2-6) in the same technology file, replace the enclosed minimum area attributes `fatTblThreshold` and `fatTblDimension` with the two attributes `fatTblMinEnclosedAreaDimension` and `fatTblMinEnclosedWidthThreshold`, to ensure correct operation of both rules.

Fat Metal Parallel Length

If two or more fat metal segments of different widths are connected in a straight line, the router considers the segments as a single long segment having a constant width for calculating the parallel lengths of nearby wires. The width of the whole “merged” fat wire segment is considered the same as the widest part. In other words, the spacing between parallel fat wires is based on the widest part of each long segment, as illustrated by the example in Figure 2-22. The minimum spacing between a merged wire and adjacent parallel wires depends on the width and length of the merged wire.

Figure 2-22 Parallel Length Determination

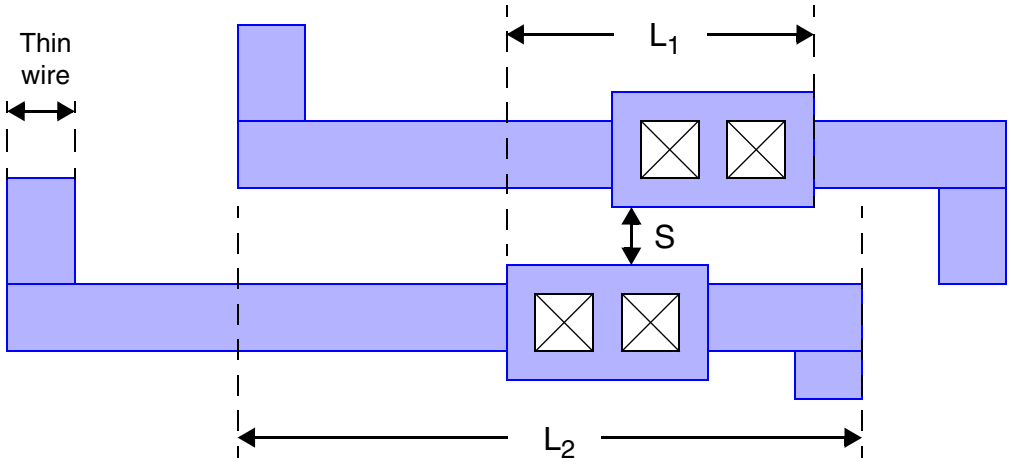


You can control the merging behavior by setting the `parallelLengthMode` attribute in the `Technology` section of the technology file. By default (`parallelLengthMode` = 0), IC Compiler merges the wire segments with fat neighboring shapes only. To merge the wire segments with all neighboring shapes, set the `parallelLengthMode` attribute to 1. For example,

```
Technology {
   parallelLengthMode = 1
}
```

Figure 2-23 demonstrates the effects of this setting. Each of the two parallel thin wires has a fat via metal enclosure. When the attribute `ParallelLengthMode` is 0 (the default), the parallel length L_1 is based on the extent of the fat enclosures only. When `ParallelLengthMode` is set to 1, the longer parallel length L_2 is based on the full extent of the merged wire.

Figure 2-23 Parallel Length Rule Example

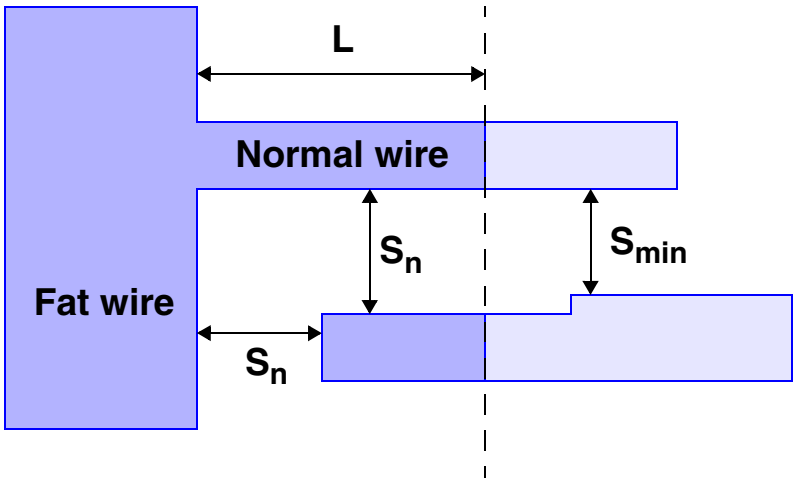


The minimum spacing between these wires depends on the parallel overlap length between them, as defined by the `FatTblParallelLength` and `fatTblSpacing` attributes described in the previous section, "Fat Metal Spacing Rule Table" on page 2-24. In Figure 2-23, the parallel length is either L_1 or L_2, depending on the `ParallelLengthMode` setting. Setting `ParallelLengthMode` to 1 results in the use of the longer parallel length L_2, which in turn can result in a larger minimum spacing requirement between the adjacent wires.

Fat Metal Extension Spacing Rule

The fat metal extension spacing rule determines when the fat metal spacing rules apply to normal wires that extend from a fat wire. Figure 2-24 shows that the portion of the extension wire within the extension threshold, L, uses the fat metal spacing rules, S_n, while the portion of the extension wire beyond the extension threshold uses normal wire spacing rules, S_{min}. You specify the extension thresholds by using the `fatTblExtensionRange` attribute.

Figure 2-24 Fat Metal Wire Having Extension Wire With Normal Width



For example,

```
Layer "M1" {
   fatTblDimension = 3
   fatTblThreshold  = (0.0, 0.501, 2.501)
   fatTblParallelLength = (0.0, 1.501, 2.501)
   fatTblExtensionRange = (0.0, 0.0, 0.8)
   fatTblSpacing  = (0.2, 0.4, 0.8,
                     0.4, 0.4, 0.8,
                     0.8, 0.8, 0.8)
```

You can control how the tool treats this rule by setting the `fatWireExtensionMode` attribute in the `Technology` section, as shown in Table 2-1.

Table 2-1 fatWireExtensionMode Attribute Values

Mode value	Description
0 (default)	The fat spacing check is performed only on extension wires connected to the fat wire and within the extension range from the fat wire edges and corners. The spacing is checked between any two wires. Projection length is not checked. See Figure 2-25 on page 2-32.
1	The fat spacing check is performed on both connected and unconnected wires within the extension range from the fat wire edges, not including fat wire corners. Spacing between both overlapped and non-overlapped wire pairs is checked. Projection lengths are also checked. See Figure 2-26 on page 2-32.

Table 2-1 fatWireExtensionMode Attribute Values (Continued)

Mode value	Description
2	The fat spacing check is performed on both connected and unconnected wires within the extension range from the fat wire edges, not including fat wire corners. Only the spacing between non-overlapped wire pairs is checked. Projection lengths are not checked. See Figure 2-27 on page 2-33.
3	The fat spacing check is performed on both connected and unconnected wires within the extension range from the fat wire edges, not including fat wire corners. Only the spacing between overlapped wire pairs is checked. Projection lengths of these wires are also checked. See Figure 2-28 on page 2-33.
4	The fat spacing check is performed only on connected wires within the extension range from the fat wire edges, not including fat wire corners. The spacing from these wires to all other wires is checked. Projection lengths are not checked. See Figure 2-29 on page 2-34.

In the following figures, the fat metal spacing applies to the shaded wire segments, based on the value of the `fatWireExtensionMode` attribute. These are the spacing values checked by the rule (which depend on the `fatWireExtensionMode` attribute setting):

- S_n: Fat spacing of metal n layer; the minimum spacing between a normal-width metal line and another metal edge when located within a specified distance (the “extension range”) of a fat wire edge.
- S_{min}: The default minimum spacing between a normal-width metal lines, which is used outside of the extension range.
- L_n: Projection length, which is the cumulative length of the projections of multiple adjacent wire segments onto the fat wire edge; see Figure 2-26 on page 2-32.

Figure 2-25 fatWireExtensionMode = 0

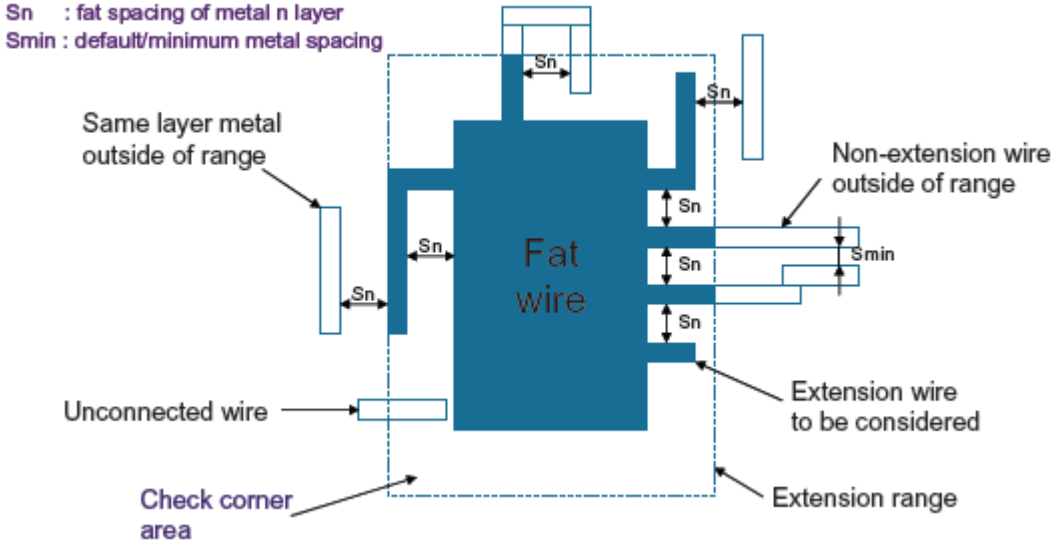


Figure 2-26 fatWireExtensionMode = 1

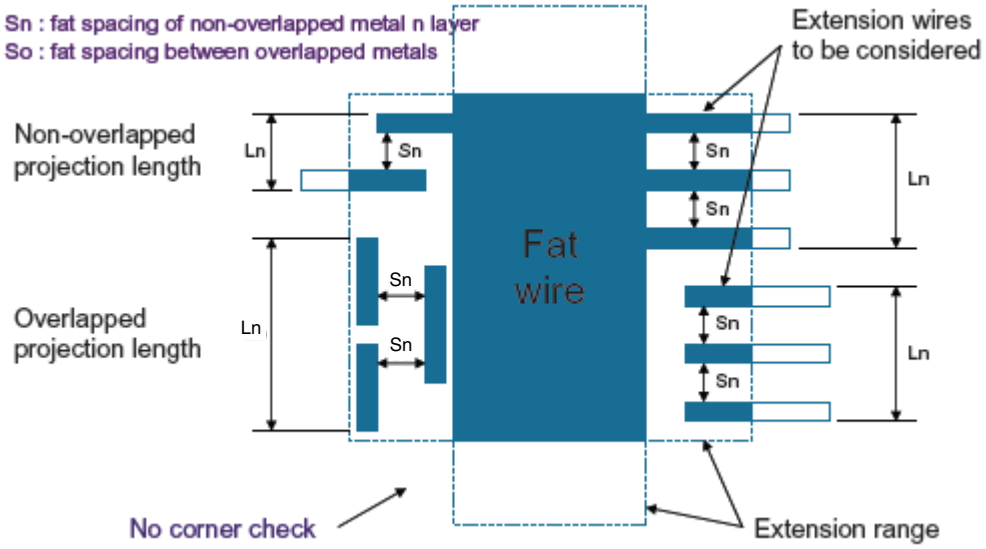


Figure 2-27 fatWireExtensionMode = 2

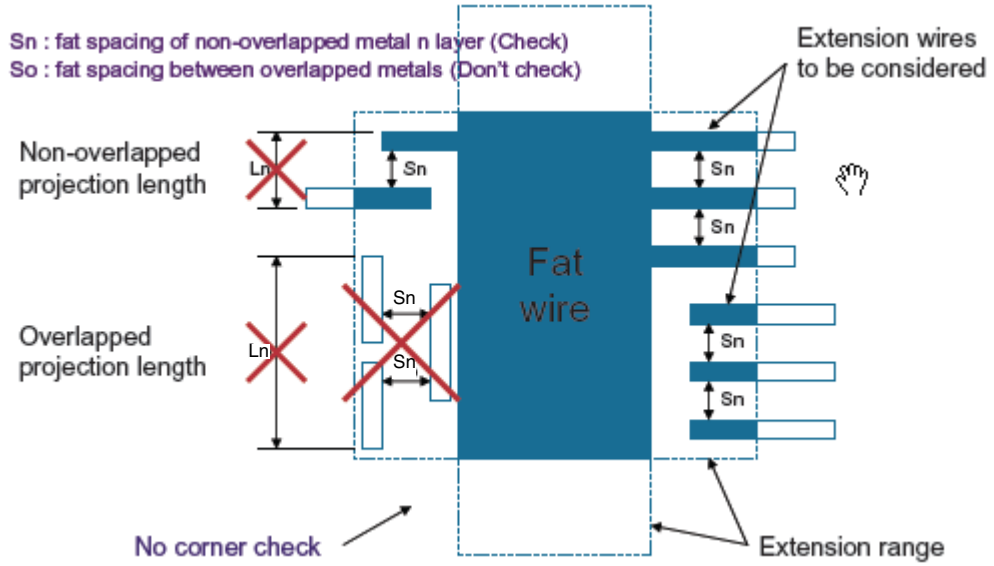


Figure 2-28 fatWireExtensionMode = 3

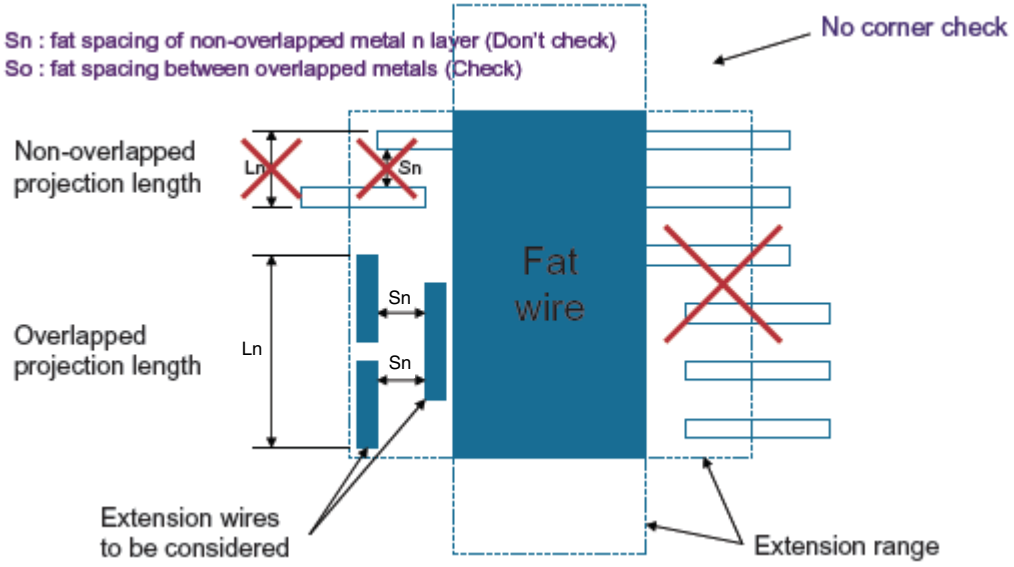
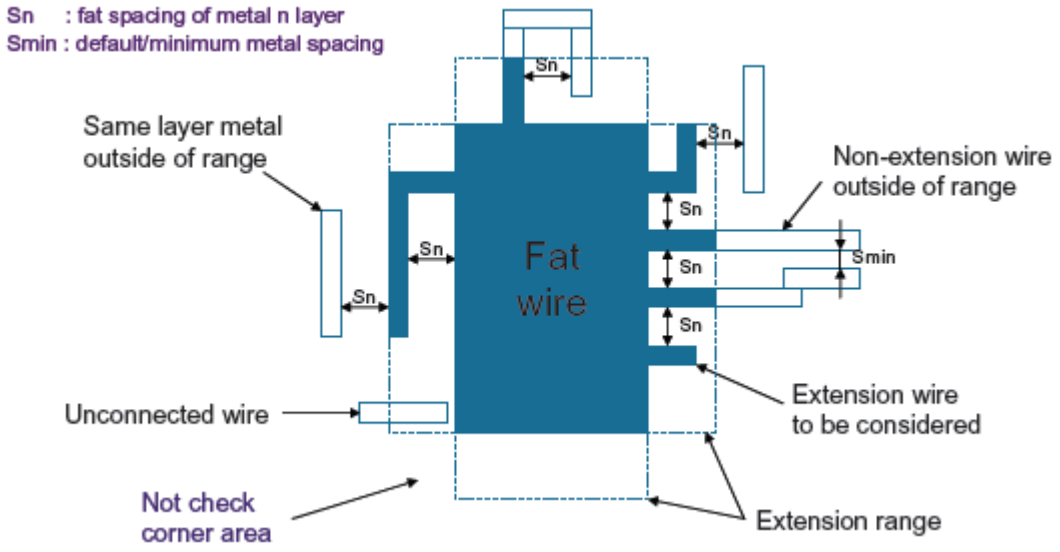


Figure 2-29 fatWireExtensionMode = 4



Neighboring Layer Fat Metal Extension Spacing Rule

You can specify a fat wire threshold and extension range. A wire on another layer that is within the defined extension range of the fat wire must meet the recommended spacing.

Use the following detail route options to specify the neighboring layer fat extension spacing rule:

```
neighboringLayerFatThreshold
neighboringLayerFatExtensionRange
neighboringLayerM1RecommendedSpacing
neighboringLayerM2RecommendedSpacing
...
neighboringLayerM12RecommendedSpacing
```

You set these options with the `set_droute_options` command. These option settings are stored with the cell.

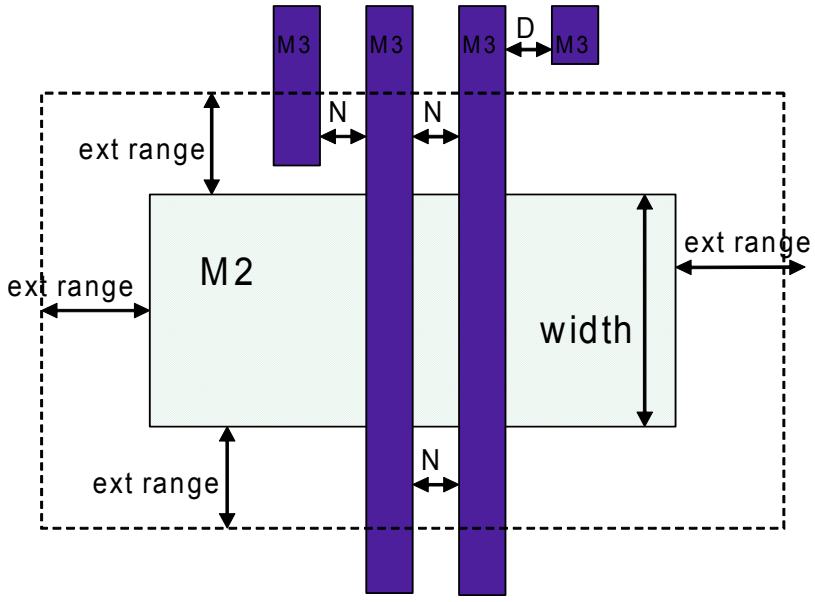
The syntax is

```
set_droute_options -name neighboringLayerFatThreshold -value 0.000
# range [0.000,100.000], default=0.000, stored in cell;
# N: neighboring layer fat threshold (applied to all layers)
```

```
set_droute_options -name neighboringLayerFatExtensionRange -value 0.000
# range [0.000,100.000], default=0.000, stored in cell;
# N: neighboring layer fat extension range (applied to all
# layers)

set_droute_options -name neighboringLayerM1RecommendedSpacing -value
0.000
# range [0.000,100.000], default=0.000, stored in cell;
# N: neighboring layer recommended spacing for M1

set_droute_options -name neighboringLayerM2RecommendedSpacing -value
0.000
# range [0.000,100.000], default=0.000, stored in cell;
# N: neighboring layer recommended spacing for M2


...


set_droute_options -name neighboringLayerM12RecommendedSpacing -value
0.000
# range [0.000,100.000], default=0.000, stored in cell;
# N: neighboring layer recommended spacing for M12
```

For example, as shown in Figure 2-30, if M2 width is greater than or equal to `neighboringLayerFatThreshold` and M3 is within `neighboringLayerFatExtensionRange`, the recommended spacing (N) applies; otherwise, the minimum spacing (D) applies.

Figure 2-30 Neighboring Layer Fat Metal Extension Spacing Rule



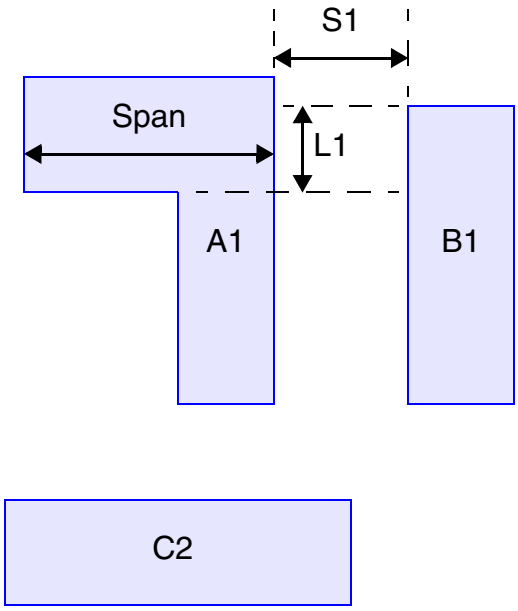
Metal Span Spacing Rule

The metal span spacing rule is similar to the fat metal spacing rule described in the section “Fat Metal Spacing Rule Table” on page 2-24. Like the fat metal spacing rule, the metal span spacing rule defines the minimum space between two parallel metal segments, subject to the parallel length between them. However, the minimum spacing value is based on the span rather than the width of the metal segments.

The span of a metal segment, unlike its width, is always measured perpendicular to the edges of the two metal segments whose spacing is under consideration. The span can be either the smaller or larger dimension of a rectangular metal segment, depending on its shape and its location with respect to the nearby metal.

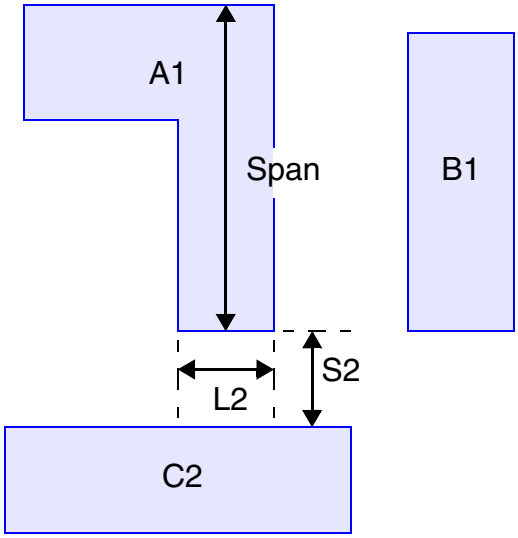
In Figure 2-31, when the metal span spacing rule considers the horizontal spacing S1 between metal A1 and metal B1, it considers the horizontal span across A1, perpendicular to the edges separating A1 and B1. It also considers the parallel length L1 where the wider top portion of metal A1 overlaps the vertical run of B1. A larger span and a larger parallel length L1 require a larger minimum spacing S1 between the two metals.

Figure 2-31 Metal Span Parameters for a Horizontal Spacing Check



In Figure 2-32, when the metal span spacing rule considers the vertical spacing S2 between metal A1 and metal C1, it considers the vertical span across A1 perpendicular to the edges separating A1 and C2. It also considers the parallel length L2 where the width of metal A1 overlaps the horizontal run of C2. A larger span and a larger parallel length L2 require a larger vertical minimum spacing S2 between the two metals.

Figure 2-32 Metal Span Parameters for a Vertical Spacing Check



The metal span spacing rule is table-based, allowing you to specify different spacing values for different ranges of span and parallel length values. The following example demonstrates usage of the rule:

```
Layer "M1" {
   spanTblDimension      = 4
   spanTblThreshold      = (0.000,0.080,0.160,0.220)
   spanTblParallelLength = (0.000,0.110,0.110,0.110)
   spanTblMinSpacing     = (0.060,0.068,0.070,0.062,
                            0.070,0.085,0.085,0.085,
                            0.071,0.085,0.088,0.088,
                            0.072,0.085,0.088,0.094)
```

The span table size is 4, so the threshold (span) and parallel length (L) attributes are each specified as a set of four values. The minimum spacing table consists of a 4-by-4 matrix corresponding to the 16 combinations of threshold and parallel length ranges.

The rule is enforced only when the span is within one of the specified ranges in the `spanTblThreshold` list and the parallel length is at least the corresponding value in the `spanTblParallelLength` list.

End-of-Line Spacing Rules

The end-of-line spacing rules define the minimum spacing between an end-of-line of a metal segment and an adjacent wire. The end-of-line spacing rules are described in the following sections:

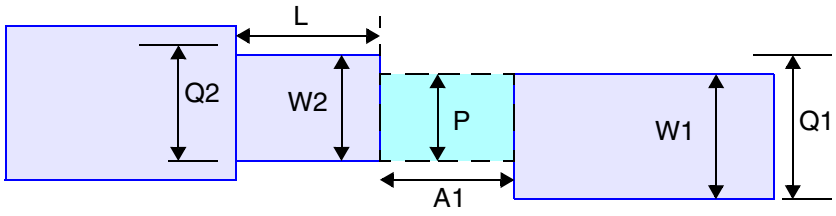
- End-of-Line to End-of-Line Spacing Rule
- One-Neighbor End-of-Line Spacing Rule
- Two-Neighbor End-of-Line Spacing Rule
- Three-Neighbor End-of-Line Spacing Rule
- Two-Sided End-of-Line Spacing Rule
- End-of-Line Spacing Rule and Stub Modes
- Enhanced Dense End-of-Line Spacing Rule
- End-of-Line to End-of-Line Spacing Rule (Classic Router)
- L-Shaped End-of-Line Spacing Rule
- End-of-Line Depth Rule

End-of-Line to End-of-Line Spacing Rule

The end-of-line to end-of-line spacing rule specifies the minimum distance between the ends of two narrow metal lines approaching each other from opposite directions.

In Figure 2-33, if the metal width W1 is less than or equal to the width threshold Q1, the metal width W2 is less than or equal to the width threshold Q2, and the parallel width overlap of the two metal lines is at least the parallel width threshold P, then the distance between the two line ends must be at least the minimum spacing value A2. If one or both metal lines widen at some distance away from the line-ends, the length of the minimum-width line-end must be at least L for this rule to apply.

Figure 2-33 End-of-Line to End-of-Line Spacing Rule




```
endOfLine1NeighborEndToEndThreshold     = Q1
endOfLine1NeighborEndToEndThreshold2    = Q2
endOfLine1NeighborEndToEndMinLength     = L
endOfLine1NeighborEndToEndParallelWidth = P
endOfLine1NeighborEndToEndMinSpacing    = A1
```

Here is an example of the syntax used for this rule:

```
Layer "M2" {
   endOfLine1NeighborEndToEndThreshold     = 0.288
   endOfLine1NeighborEndToEndThreshold2    = 0.270
   endOfLine1NeighborEndToEndMinLength     = 0.300
   endOfLine1NeighborEndToEndParallelWidth = 0.144
   endOfLine1NeighborEndToEndMinSpacing    = 0.104
}
```

In this example, for layer M2, when two lines, one having a width of no more than 0.288 and another with a width no more than 0.270, approach from opposite directions, the two line ends must have a spacing of at least 0.104. For wires that vary in width, the lengths of the minimum-width line-ends must be at least 0.300 for this rule to apply.

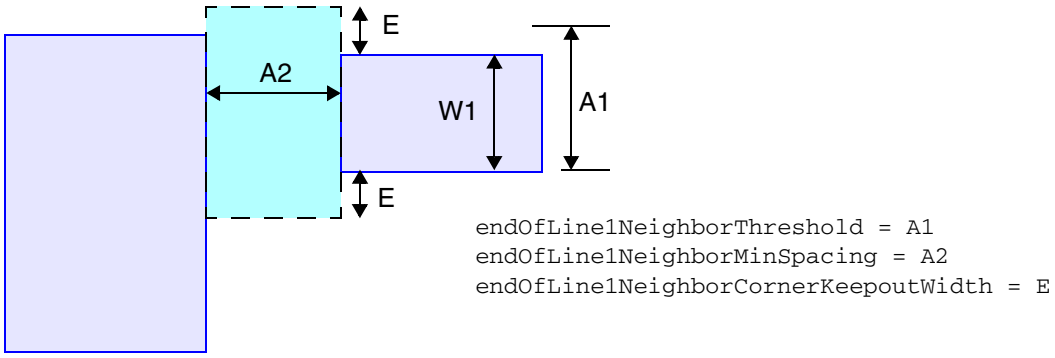
Note:

This rule is supported only by Zroute. For a similar rule supported by the classic router, see “End-of-Line to End-of-Line Spacing Rule (Classic Router)” on page 2-52.

One-Neighbor End-of-Line Spacing Rule

The one-neighbor end-of-line spacing rule specifies the minimum distance between the end of a narrow metal line and another metal. In Figure 2-34, if the metal width W1 is less than or equal to the width threshold A1, then the distance between the line end and the other metal must be at least the minimum spacing value A2. The two metals must be outside of the dashed rectangle extending outward from the line-end corners by the keepout width E.

Figure 2-34 End-of-Line to End-of-Line Spacing Rule



Here is an example of the syntax used for this rule:

```
Layer "M2" {
   endOfLine1NeighborThreshold = 0.288
   endOfLine1NeighborMinSpacing = 0.140
   endOfLine1NeighborCornerKeepoutWidth = 0.020
}
```

In this example, for layer M2, when a metal line having a width of less than 0.288 approaches another metal, the spacing between them must be at least 0.140, including a keepout region extending 0.020 away from the line-end corners.

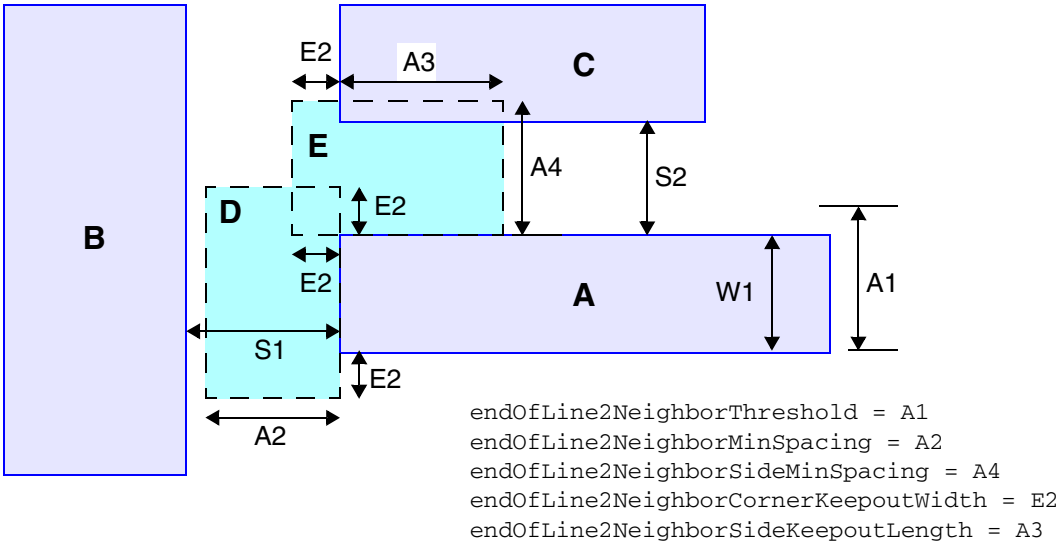
Note:

This rule is supported only by Zroute. For a similar rule supported by the classic router, see "Stub Mode 1: Single-Edge Spacing at Metal End" on page 2-47.

Two-Neighbor End-of-Line Spacing Rule

The two-neighbor end-of-line spacing rule specifies the minimum distance between the end of a narrow metal line and a neighboring metal and from the side of the same narrow metal line to another neighboring metal. In Figure 2-35, if the metal width W1 is less than or equal to the width threshold A1, then the distance between the line end and the first other metal S1 must be at least the minimum spacing value A2, or the distance between the side of the narrow metal line and the second other metal S2 must be at least the minimum side spacing value A4.

Figure 2-35 Two-Neighbor End-of-Line Spacing Rule



In the figure, either metal B must be outside of the dashed rectangle D, or metal C must be outside of the dashed rectangle E. In this case, metal B is outside of dashed rectangle D, so the layout passes the rule. The dashed areas extend outward from the line-end corners by the corner keepout width E2. The side keepout area E has a length of A3 plus extension E2.

Here is an example of the syntax used for this rule:

```
Layer "M2" {
   endOfLine2NeighborThreshold = 0.070
   endOfLine2NeighborMinSpacing = 0.075
   endOfLine2NeighborSideMinSpacing = 0.080
   endOfLine2NeighborCornerKeepoutWidth = 0.030
   endOfLine2NeighborSideKeepoutLength = 0.075
}
```

In this example, for layer M2, when a metal line having a width of less than 0.070 has neighboring metal at the end and at the side near the end, the spacing to the neighboring metal at the end must be at least 0.075 or the spacing to the neighboring metal at the side must be at least 0.080. The keepout region extends 0.030 away from the line-end corners and 0.080 from the corner along the side.

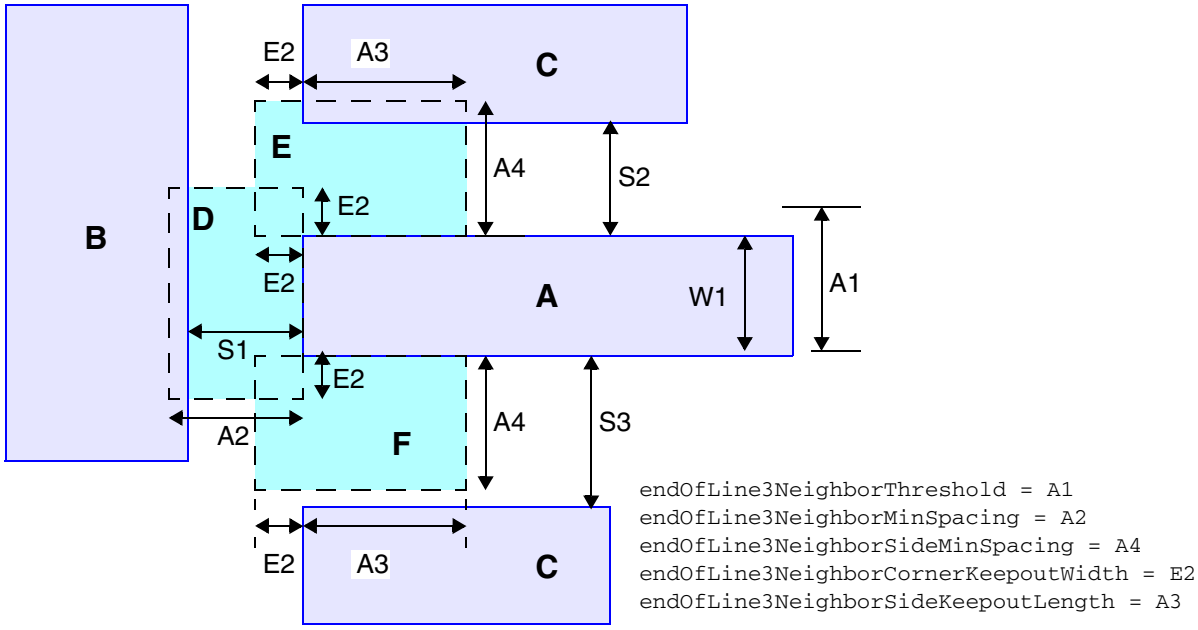
Note:

This rule is supported only by Zroute. For a similar rule supported by the classic router, see “Stub Mode 4: Two-Edge Spacing With Connected Metal Exclusion” on page 2-51.

Three-Neighbor End-of-Line Spacing Rule

The three-neighbor end-of-line spacing rule specifies the minimum distance between the end of a narrow metal line and a neighboring metal, and from the two sides of the same narrow metal line to two other neighboring metals. In Figure 2-36, if the metal width W1 is less than or equal to the width threshold A1, then the distance between the line end and the metal S1 must be at least the minimum spacing value A2, or if not, the distance between at least one of the two sides of the narrow metal line and the neighboring metals, S2 or S3, must be at least the minimum side spacing value A4.

Figure 2-36 Three-Neighbor End-of-Line Spacing Rule



In the figure, either metal B must be outside of the dashed rectangle D, or if not, either metal C must be outside of the dashed rectangle E or metal C must be outside of the dashed rectangle F. In this case, metal C is outside of dashed rectangle F, so the layout passes the rule. The dashed areas extend outward from the line-end corners by the corner keepout width E2. The side keepout areas E and F each have a length of A3 plus extension E2.

Here is an example of the syntax used for this rule:

```
Layer "M2" {
   endOfLine3NeighborThreshold = 0.288
   endOfLine3NeighborMinSpacing = 0.14
   endOfLine3NeighborSideMinSpacing = 0.16
   endOfLine3NeighborCornerKeepoutWidth = 0.02
   endOfLine3NeighborSideKeepoutLength = 0.27
}
```

In this example, for layer M2, when a metal line having a width of less than 0.288 has neighboring metal at the end and at two sides near the end, the spacing to the neighboring metal at the end must be at least 0.14 or the spacing to the neighboring metal on at least one of the two sides must be at least 0.16. The two side keepout regions extend 0.02 away from the line-end corners and 0.16 from the corner along the sides.

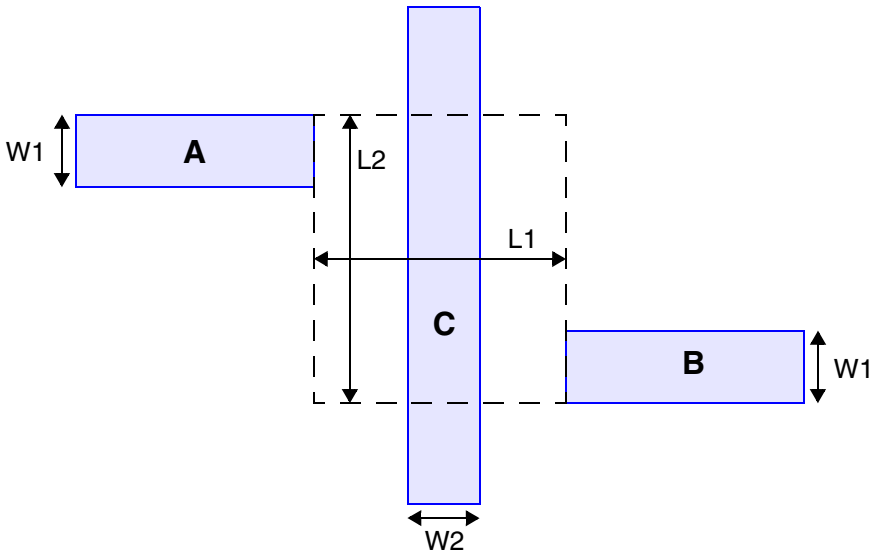
Note:

This rule is supported only by Zroute. For a similar rule supported by the classic router, see “Stub Mode 3: Three-Edge Spacing at Metal End” on page 2-50.

Two-Sided End-of-Line Spacing Rule

The two-sided end-of-line spacing rule specifies the minimum distance between two narrow end-of-line wires approaching a third narrow wire from both sides at 90 degrees. In Figure 2-37, the two narrow wires A and B, both having a width less than or equal to W1, approach a third narrow wire C having a width less than W2, from opposite directions. The ends of wires A and B must stay outside of the rectangle measuring L1 by L2 and centered on wire C.

Figure 2-37 Two-Sided End-of-Line Spacing Rule



To invoke this rule in Zroute, use the following syntax in the `Layer` section:

```
tJunctionStubWireMaxThreshold   = W1
tJunctionOrthoWireMaxThreshold  = W2
tJunctionStubKeepoutMinSpacing  = L2
tJunctionStubKeepoutMinWidth    = L1
```

For example,

```
tJunctionStubWireMaxThreshold   = 0.20
tJunctionOrthoWireMaxThreshold  = 0.22
tJunctionStubKeepoutMinSpacing  = 0.80
tJunctionStubKeepoutMinWidth    = 0.70
```

Note:

This rule is supported only by Zroute; the classic router does not support this rule.

Figure 2-38 shows two examples of violations. When the two end-of-line wires approach the third wire directly opposite from each other or offset by as much as L2 as measured from their outer edges, both ends must stay outside of the box measuring L1 in length. The black rectangle represents the overlap of the two approaching wires with the central wire.

Figure 2-38 Two-Sided End-of-Line Spacing Violation Examples

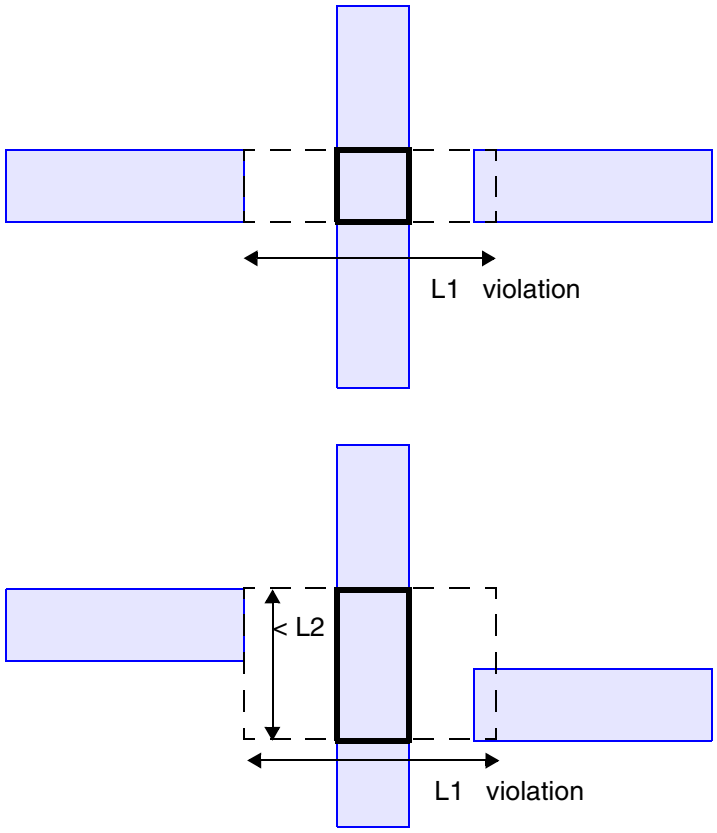
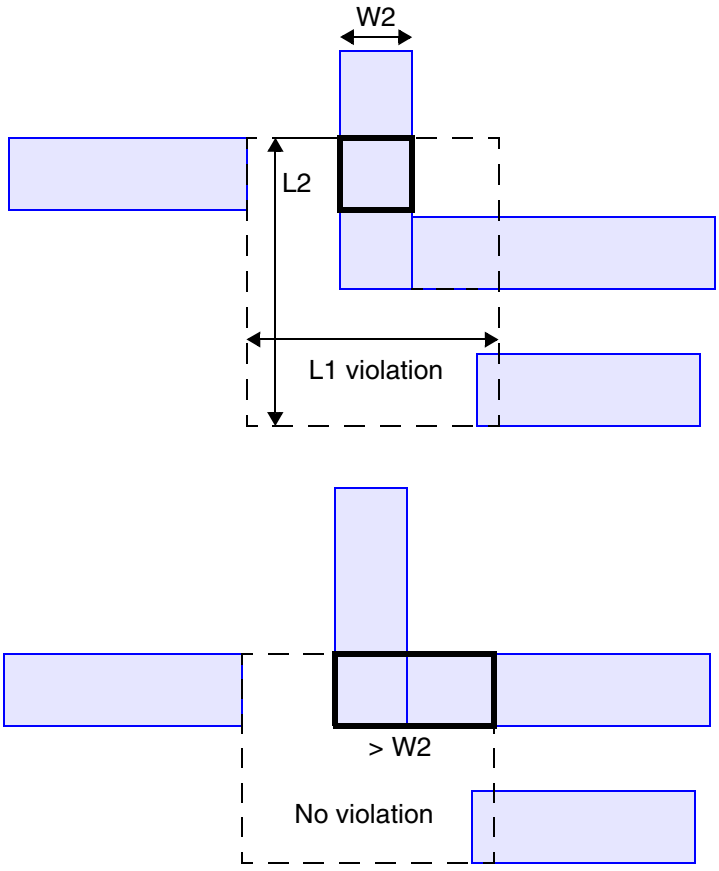


Figure 2-39 shows two examples of end-of-line wires approaching an L-shaped wire. The black rectangle represents the overlap of the two approaching wires with the L-shaped wire. The first example is a violation. However, the second example is not a violation because the overlap applies to the long section of the L-shaped wire. The L-shaped wire's width is considered to be greater than W2, so it is not a narrow wire for this application of the rule, and the rule does not apply.

Figure 2-39 Two-Sided End-of-Line Spacing Violation at “L”



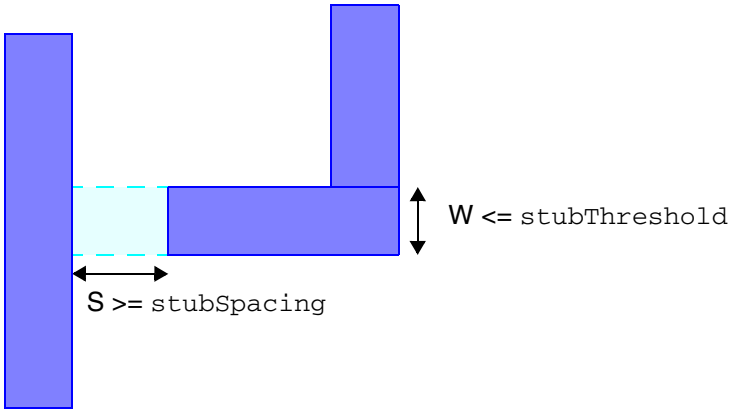
End-of-Line Spacing Rule and Stub Modes

The end-of-line spacing rule defines a larger minimum spacing requirement where the end of a wire is perpendicular to another wire. You define this rule by setting the following attributes in a metal `Layer` section of the technology file:

- The `stubSpacing` attribute defines the minimum spacing between the end-of-line metal segment and its adjacent wire.
- The `stubThreshold` attribute defines the maximum width of the end-of-line metal segment to which this rule applies. If the wire is wider than the `stubThreshold` value, the default minimum spacing applies.

Figure 2-40 illustrates the application of this rule. The rule applies only to the lightly shaded area between the wire end and the adjacent wire. When the end-of-line wire width W is less than `stubThreshold`, the spacing S must be at least `stubSpacing`.

Figure 2-40 End-of-Line Spacing Rule



Depending on the foundry process specification, you should specify a value of 1, 2, 3, or 4 for the `stubMode` attribute in the `Technology` section of the technology file. The stub mode determines in detail the types of checking performed in different end-of-line situations.

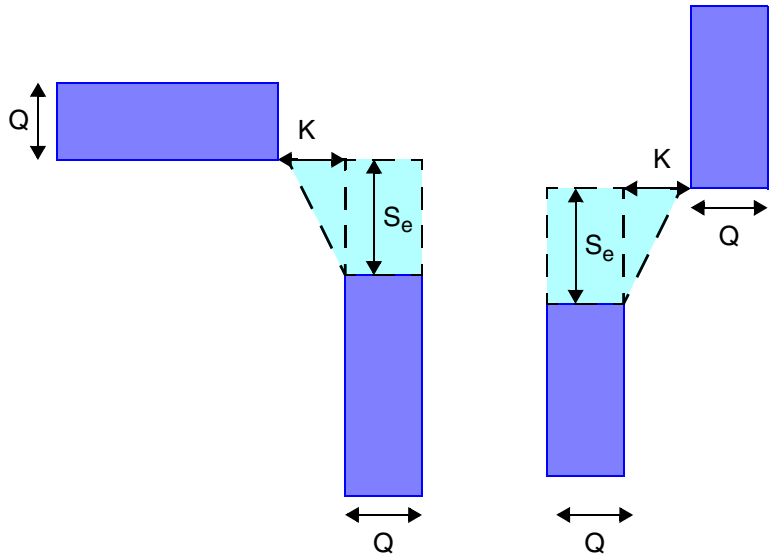
For example,

```
Technology {
   stubMode = 1
}

Layer "M1"  {
   stubSpacing    = 0.14
   stubThreshold  = 0.20
}
```

Stub Mode 1: Single-Edge Spacing at Metal End

With stubMode set to 1, the minimum spacing between two narrow metal end-of-lines is S_e (`stubSpacing`) when the metal widths are both less than or equal to Q (`stubThreshold`). This rule applies to metal end-of-lines that are offset by as much as the distance K (`endOfLineCornerKeepoutWidth`). In each of the two examples shown in Figure 2-41, the upper blue metal line must stay outside of the lightly shaded regions between it and the lower metal line.

Figure 2-41 Stub Mode 1



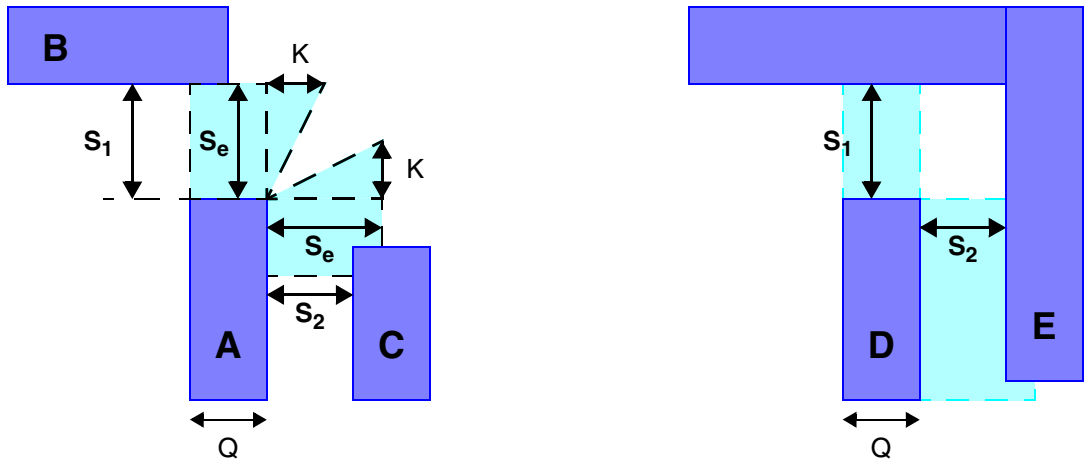
This is the syntax for specifying the end-of-line rule in stub mode 1:

```
Technology {
  stubMode = 1
  }

Layer "Metal1" {
  stubSpacing = Se
  stubThreshold = Q
  endOfLineCornerKeepoutWidth = K
  }
```

Stub Mode 2: Two-Edge Spacing at Metal End

With stubMode set to 2, when two adjacent edges of a narrow metal end-of-line are close to two neighboring metal edges, or when any one short edge is close to an inside corner between two neighboring edges, then at least one of the two spacing values must be greater than the specified stub spacing value. Two examples are illustrated in Figure 2-42.

Figure 2-42 Stub Mode 2



In the first example, the top and right edges of metal A are close to the edges of metal B and metal C. The width of metal A is less than or equal to Q (`stubThreshold`). Either the spacing S_1 or the spacing S_2 must be at least S_e (`stubSpacing`). One of the two spacings (metal C in this example) can be less than S_e as long as it meets the general metal-to-metal spacing requirement and the other spacing (metal B in this example) meets the S_e spacing requirement. Metal that is offset by as much as K (`endOfLineCornerKeepoutWidth`) is still considered by this rule.

In the second example, the width of metal D is less than or equal to Q (`stubThreshold`) and the metal is close to the inside corner of metal E. Of the two spacing values S_1 and S_2, at least one of them must be at least S_e (`stubSpacing`). In this example, S_1 meets this requirement. S_2 is less than S_e but still meets the general metal-to-metal minimum spacing requirement.

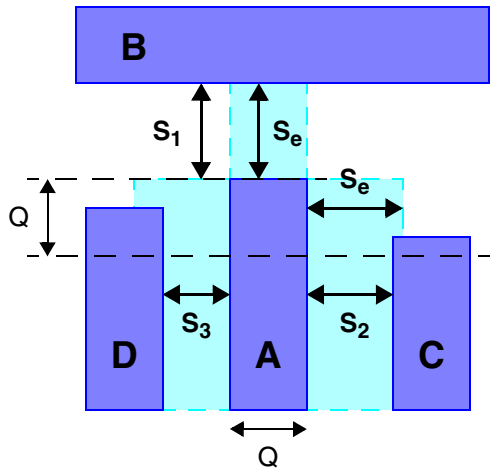
This is the syntax for specifying the end-of-line rule in stub mode 2:

```
Technology {
  stubMode = 2
  }

Layer "Metal1" {
  stubSpacing = Se
  stubThreshold = Q
  endOfLineCornerKeepoutWidth = K
  }
```

Stub Mode 3: Three-Edge Spacing at Metal End

With stubMode set to 3, when three adjacent edges of a narrow metal end-of-line are close to three neighboring metal edges, and two of the neighboring metals are end-of-lines ending within the stub threshold, then at least one of the three spacing values must be greater than the specified stub spacing value. An example is shown in Figure 2-43.

Figure 2-43 Stub Mode 3



In this example, metal A has a width less than or equal to Q (`stubThreshold`) and has three neighboring metals, B, C, and D. The ends of metals C and D are offset from the end of metal A by no more than Q. At least one of the three metal-to-metal spacing values S_1, S_2, and S_3 must be at least the stub spacing S_e (`stubSpacing`). In this example, spacing S_1 meets this requirement. S_2 and S_3 must still meet the general metal-to-metal minimum spacing requirement.

This is the syntax for specifying the end-of-line rule in stub mode 3:

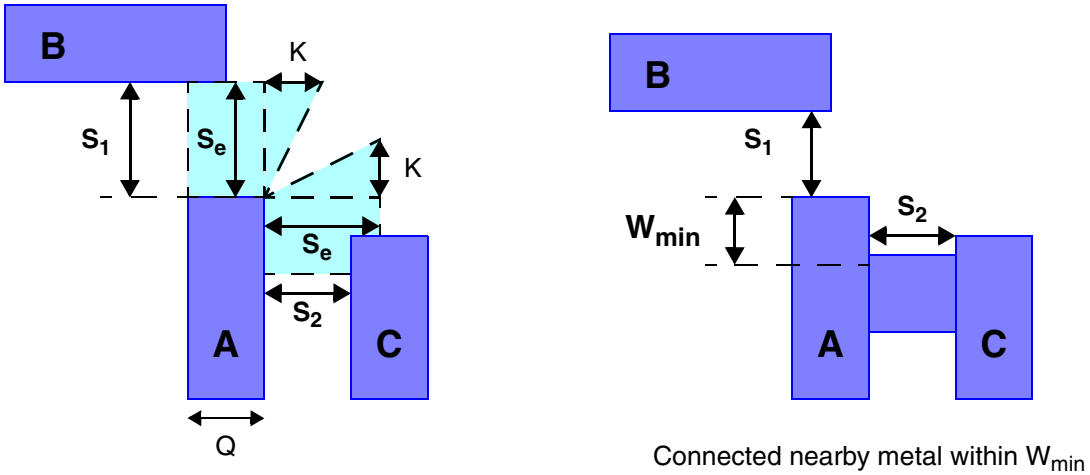
```
Technology {
  stubMode = 3
  }

Layer "Metal1" {
  stubSpacing = Se
  stubThreshold = Q
  }
```

Stub Mode 4: Two-Edge Spacing With Connected Metal Exclusion

Stub mode 4 is similar to stub mode 2. The only difference is that the rule does not apply to the case where the nearby metal is connected with a jog or displacement of less than the default minimum width, W_{min} (`minWidth`).

In the first example shown in Figure 2-44, there is no connection between metal A and metal C, so stub mode 4 is the same as stub mode 2. However, in the second example, metal A is connected to the nearby metal and the size of the jog is less than W_{min}, so the rule does not apply. Even if S_1 and S_2 are both less than S_e, it is not a violation of the stub mode 4 rule, whereas it would be a violation with the stub mode 2 rule. If the size of the jog is greater than W_{min}, then the rule still applies and stub mode 4 still works like stub mode 2.

Figure 2-44 Stub Mode 4



Unconnected nearby metal C

Connected nearby metal within W_{min}
Rule does not apply

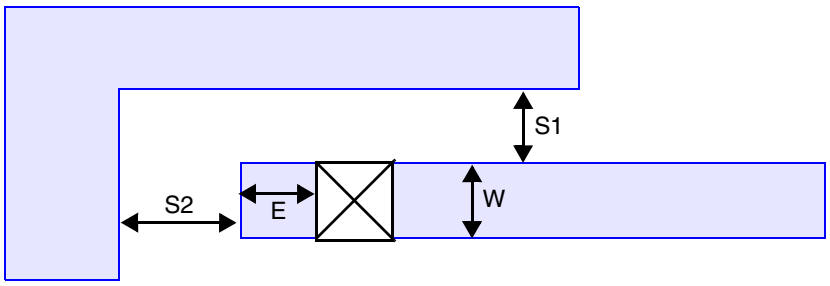
This is the syntax for specifying the end-of-line rule in stub mode 4:

```
Technology {
  stubMode = 4
  }

Layer "Metal1" {
  minWidth = Wmin
  stubSpacing = Se
  stubThreshold = Q
  endOfLineCornerKeepoutWidth = K
  }
```

Enhanced Dense End-of-Line Spacing Rule

The enhanced dense end-of-line spacing rule defines the minimum end-of-line spacing for wire ends with a via dropped to the upper or lower via layer when the width (W) of the wire is less than the stub threshold as defined in the `stubThreshold` attribute in the associated `Layer` section in the technology library. Figure 2-45 shows the parameters associated with this rule.

Figure 2-45 Enhanced Dense End-of-Line Spacing Rule



This rule is defined in the `DesignRule` section of the technology file. The minimum spacing requirement depends on the width of the via enclosure (E). The enclosure width thresholds are defined in the `endOfLineEncTbl` attribute and the end-of-line minimum spacing requirements (S2) are specified in the `endOfLineEncSpacingTbl` attribute. The S1 spacing requirement is specified as the largest value in the `endOfLineEncSpacingTbl` attribute. For example,

```
DesignRule {
   layer1 = "M2"
   layer2 = "VIA1"
   endOfLineEncTblSize = 5
   endOfLineEncSpacingTbl = (0.1, 0.105, 0.11, 0.115, 0.12)
   endOfLineEncTbl = (0.05, 0.045, 0.04, 0.035, 0.03)
}
```

In this example, S1 spacing requirement is 0.12.

End-of-Line to End-of-Line Spacing Rule (Classic Router)

Note:

This rule is supported only by the classic router. For an equivalent rule supported by Zroute, see “End-of-Line to End-of-Line Spacing Rule” on page 2-38.

Use the `stubToStubSpacing` and `endOfLineCornerKeepoutWidth` attributes as well as the `stubSpacing` attribute to specify the end-of-line to end-of-line spacing rule, also known as the tip-to-tip spacing rule. Define these attributes in a metal `Layer` section of the technology file.

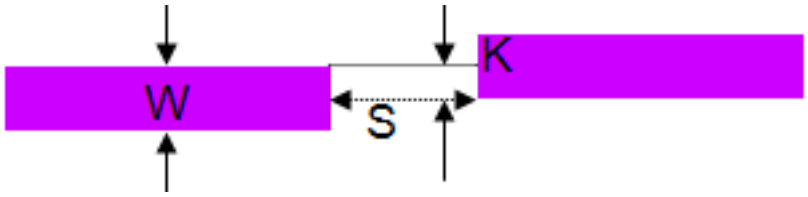
The `stubToStubSpacing` specifies the spacing between two end-of-line wire segments. The `endOfLineCornerKeepoutWidth` specifies the parallel projection between the two end-of-line edges, defining a keepout region.

Also, you must specify a value of 1 for the `stubMode` attribute in the `Technology` section of the technology file.

For example,

```
Technology {
   stubMode    = 1
}
Layer  "M2" {
   layerNumber                    = 3
   maskName                       = "metal2"
   stubSpacing                    = 0.088
   stubToStubSpacing              = 0.104
   stubThreshold                  = 0.288
   endOfLineCornerKeepoutWidth    = -0.002
}
```

The following figure shows how this syntax is interpreted:



- When the wire Mx width (W) is less than 0.288, the minimum space (S) between an end-of-line wire and an end-of-line wire (tip-to-tip spacing) must be greater than or equal to 0.104.
- The wire must have ends overlapping projection (K). In this example, K is greater than or equal to 0.002.

L-Shaped End-of-Line Spacing Rule

L-shaped end-of-line spacing is applied when the end-of-line segment is part of an L-shape geometry and its length meets the given length threshold. The L-shape definition includes any shapes that have a partial L shape, such as a T shape.

Use the `stubLengthThreshold` attribute as well as the `stubSpacing` and `stubThreshold` attributes to specify this rule. Define these attributes in a metal `Layer` section of the technology file.

Also, you must specify a value of 1 for the `stubMode` attribute in the `Technology` section of the technology file.

The `stubSpacing` defines the end-of-line edge or corner of the metal whose edge width is less than or equal to the given `stubThreshold`. An L shape with an adjacent segment on the same layer is required, and at least one of the side edge lengths must be less than or equal to the given `stubLengthThreshold`. Otherwise, the default minimum spacing specified with `minSpacing` applies.

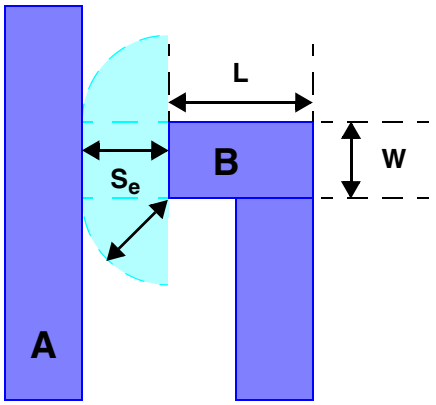
The L-shape rule is applied to the end-of-line segment edge as well as to the diagonal corner spacing, provided that the L-shape geometry and its length meet the given length threshold.

For example,

```
Technology {
   stubMode = 1
}
Layer "M7" {
   minSpacing           = 0.20
   stubSpacing          = 0.22
   stubThreshold        = 0.24
   stubLengthThreshold  = 0.28
}
```

With the foregoing rule definitions, in Figure 2-46, the L-shaped end-of-line spacing rule applies only when the length of the segment L is less than 0.28 (`stubLengthThreshold`) and the width of the segment W is less than 0.24 (`stubThreshold`). If both of these conditions are true, then the minimum spacing is S_e (`minSpacing`); otherwise, it is the default metal-to-metal minimum spacing.

Figure 2-46 L-Shaped End-of-Line Spacing Rule



This rule is typically set to be less restrictive than the ordinary end-of-line minimum spacing rule, thus conserving routing resources for the L-shaped case versus the normal case.

End-of-Line Depth Rule

Note:

This rule is supported only by the classic router; Zroute does not support this rule.

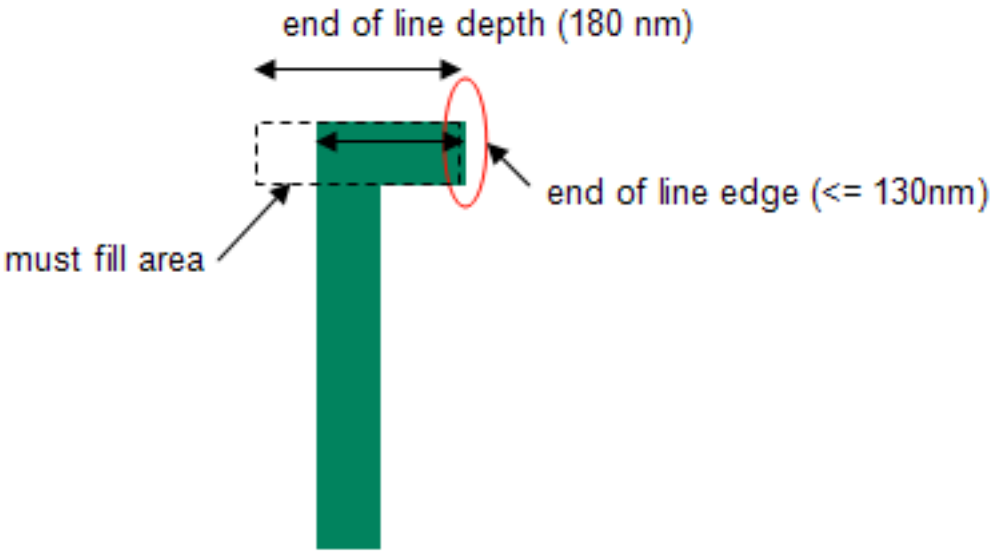
Short segments near line ends cannot always be resolved on the wafer and can create problems for optical control systems. To avoid this situation, use the end-of-line depth rule. This rule specifies that the distance from the end-of-line edge to the opposite internal edge is to be greater than or equal to the value you define with the `stubMinLength` attribute for a stub with a line end width less than or equal to the value you define with the `stubThreshold` attribute. Define these attributes in a metal `Layer` section of the technology file.

For example,

```
Layer "Metal1" {
   stubThreshold  = 0.130
   stubMinLength  = 0.180
}
```

Figure 2-47 shows the parameters associated with the end-of-line depth rule.

Figure 2-47 End-of-Line Depth Rule



- The `stubThreshold` value defines the end-of-line edge width. The end-of-line edge is any edge less than or equal to 0.130 connected to two convex vertices.
- The `stubMinLength` value defines a “must fill” area, starting from the end-of-line edge. If the must fill area is not filled, a violation is reported.

Via Spacing Rules

The via spacing rules are described in the following sections:

- Adjacent Via Rule
- Enclosed Via Spacing Rule
- Isolated Via Rule

Adjacent Via Rule

The adjacent via rule specifies the maximum number of adjacent vias within a specified range of a via.

You define the adjacent via rule by setting the following attributes in a via `Layer` section of the technology file:

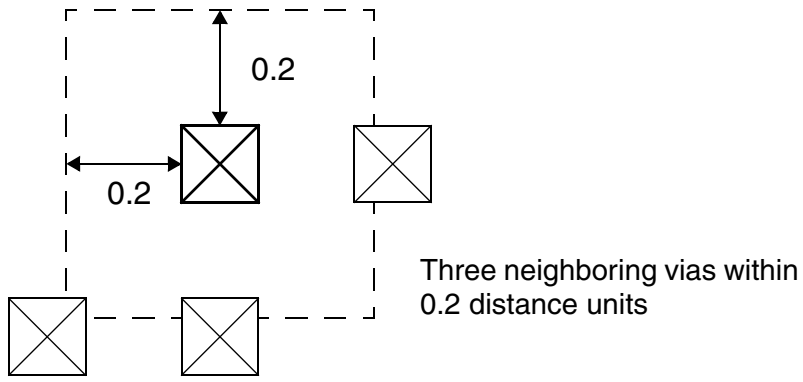
- The `maxNumAdjacentCut` attribute defines the maximum number of adjacent vias.
- The `adjacentCutRange` attribute defines the range within which to check for adjacent vias.

For example,

```
layer   "VIA1" {
    maxNumAdjacentCut      = 2
    adjacentCutRange      = 0.2
}
```

In Figure 2-48, there are more than two adjacent vias within the specified distance range, thereby triggering a rule violation. It does not matter whether the adjacent vias belong to the same net or a different net.

Figure 2-48 Adjacent Via Rule



Enclosed Via Spacing Rule

A via surrounded by at least a specified number of vias within a specified distance range is called an enclosed via. You can specify the minimum distance between enclosed vias and the minimum distance between an enclosed via and a neighboring via. A neighboring via is a nearby via that is not an enclosed via.

Use the following attributes in a via `Layer` section of the technology file to specify the enclosed via spacing rule:

`enclosedCutNumNeighbor`

Specifies the minimum number of neighboring vias allowed for defining an enclosed via.

`enclosedCutNeighborRange`

Specifies the range of neighboring vias for defining an enclosed via.

`enclosedCutMinSpacing`

The minimum spacing between two enclosed vias.

`enclosedCutToNeighborMinSpacing`

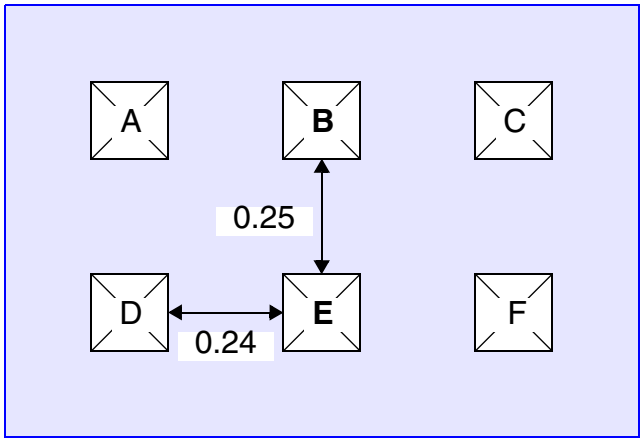
Specifies the minimum spacing allowed between an enclosed via and surrounding vias.

For example,

```
Layer   "V3"  {
    enclosedCutNumNeighbor             =   3
    enclosedCutNeighborRange           =   0.4
    enclosedCutMinSpacing              =   0.25
    enclosedCutToNeighborMinSpacing    =   0.24
}
```

In this example, a via is defined to be an enclosed via when it has three or more nearby vias within a distance of 0.4. The minimum allowed spacing between two enclosed vias is 0.25. The minimum allowed spacing between an enclosed via and a neighboring, non-enclosed via is 0.24. In Figure 2-49, vias B and E in the 3-by-2 via array are enclosed vias because they each are surrounded by three nearby vias, whereas vias A, C, D, and F are not enclosed vias. The minimum spacing between the two enclosed vias is 0.25, whereas the minimum spacing between an enclosed and neighboring, non-enclosed via is 0.24.

Figure 2-49 Enclosed Via Spacing Rule Example



Isolated Via Rule

The isolated via rule defines the maximum distance between vias. It is specified by setting detail route options. You set these options with the `set_droute_options` command. These option settings are stored with the cell.

An isolated via is a via that does not have neighboring vias close enough to meet the requirements of the technology. To pass this rule, a via must meet at least one of the following two requirements:

- An adjacent via is located within the distance in microns that is specified in the `isolatedViaSpacing` detail route option.

  The value of this option must be between 0.0 and 20.0. If you do not set this option, a distance of 1.0 micron is used.

- Adjacent vias exist in all four surrounding quadrants within the distance in microns that is specified in the `isolatedViaQuadrantSpacing` detail route option.

  The value of this option must be between 0.0 and 50.0, and it must be greater than the value specified for `isolatedViaSpacing`. If you do not set this option, a distance of 10.0 microns is used.

The values that you set for these detail route options are stored with the design when you save the design in Milkyway format.

To check for isolated via violations in your design, run the `report_isolated_via` command. The tool measures the spacing between vias as follows:

- If the vias are not aligned, the tool measures corner-to-corner spacing.
- If the vias are aligned, the tool measures edge-to-edge spacing.

The tool checks the isolated via spacing first and then checks the isolated via quadrant spacing. If a via fails both tests, it is flagged as an isolated via violation.

To fix isolated via violations, run the `fix_isolated_via` command. By default, the tool fixes the isolated via violations for all unfixed signal, clock, power, and ground nets by inserting a hang-on via (a via that is connected to only one metal layer) on the net.

To check for and fix isolated via violations on fixed nets, set the `droute_checkFixedDRC` variable to 1. In this case, the violations are fixed by adding a via to a unfixed routing segment.

To fix isolated via locations that occur on clock nets without touching the clock net itself, set the `droute_fixIsoViaTouchClock` variable to 0. To fix isolated via violations on power and ground nets without touching the power or ground net itself, set the `droute_fixIsoViaTouchPG` variable to 0.

Fat Metal Contact Rules

The fat metal contact rules are described in the following sections:

- Fat Metal Contact Rule
- Fat Metal Extension Contact Rule
- Fat Poly Contact Rule

Fat Metal Contact Rule

The fat metal contact rule specifies the minimum number of vias required to connect between two metal layers when either of the metal segments is a fat wire. You can specify a fat metal contact rule by using either a one-dimensional or two-dimensional table rule. When you use a one-dimensional table rule, the thresholds are determined by the maximum width of the upper and lower layer metal segments. When you use a two-dimensional rule, you specify separate thresholds for the upper and lower metal layers. You can define a fat metal extension contact rule by specifying an extension threshold for the fat metal contact rule.

One-Dimensional Table Rule

To define a one-dimensional fat metal contact rule, specify the following attributes in a via `Layer` section of the technology file:

- `fatTblDimension`

  Specifies the size of the one-dimensional fat table.

- `fatTblThreshold`

  Specifies the thresholds used to determine which rules apply, based on the maximum width of the metal segments on the upper and lower metal layers. The width is the smaller of the x-length or y-length of the metal segment. You must specify *n* values, where *n* is the table dimension.

- `fatTblFatContactNumber`

  Specifies the contact code numbers for the fat wires. You must specify *n* values, where *n* is the table dimension.

- `fatTblFatContactMinCuts`

  Specifies the minimum number of cuts. You must specify *n* values, where *n* is the table dimension.

In the following example, if either of the metal widths connected by the VIA1 layer is greater than or equal to 0.155, then a via whose contactCode value is defined as 21 will be used and at least two cuts of this via will be necessary.

```
layer   "VIA1" {

Dimension     = 3
    fatTblThreshold     = (0, 0.155, 1.605)
    fatTblFatContactNumber  = (1, 21, 31)
    fatTblFatContactMinCuts = (1, 2, 4)
}
```

Two-Dimensional Table Rule

Both the power and ground routing and the detail routing operations honor the two-dimensional table rule. A rule violation is flagged as a DRC error.

To define a two-dimensional fat metal contact rule, specify the following attributes in a via `Layer` section of the technology file:

- `fatTblDimension`

  Specifies the size of the two-dimensional fat table.

- `fatTblThreshold`

  Specifies the thresholds used to determine which rules apply, based on the width of the metal segment on the lower layer. The width is the smaller of the x-length or y-length of the metal segment. You must specify *n* values, where *n* is the table dimension.

- `fatTblThreshold2`

  Specifies the thresholds used to determine which rules apply, based on the width of the metal segment on the upper layer. The width is the smaller of the x-length or y-length of the metal segment. You must specify *n* values, where *n* is the table dimension.

- `fat2DTblFatContactNumber`

  Specifies the contact code numbers used to select vias from the library. You must specify an *n* x *n* table, where *n* is the table dimension.

- `fat2DTblFatContactMinCuts`

  Specifies the minimum number of cuts. You must specify an *n* x *n* table, where *n* is the table dimension.

Note:

In the two-dimensional fat contact tables, the horizontal index is for the lower metal layer (`fatTblThreshold`) and the vertical index is for the upper metal layer (`fatTblThreshold2`).

In the following example, if the lower metal width value is greater than or equal to 0.421 and less than 0.701, the tool considers that value as element 2 of `fatTblThreshold`. If the upper metal width value is greater than or equal to 0.501 and less than 0.781, then the tool considers that value as element 3 of `fatTblThreshold2`. Consequently, the tool picks up the (2,3) element value from `fat2DTblFatContactNumber`, which is 18, and the (2,3) element from `fat2DTblFatContactMinCuts`, which is 3. So, IC Compiler uses the via whose `contactCode` value is 18 and uses at least three cuts of the via to connect the lower and upper metal layers.

```
"Layer "VIA23" {
    fatTblDimension = 4
    /* Threshold for lower layer metal width */
    fatTblThreshold = (0.0, 0.421, 0.701, 0.981)
    /* Threshold for upper layer metal width */
    fatTblThreshold2 = (0.0, 0.221, 0.501, 0.781)
    fat2DTblFatContactNumber = (1, 18, 17, 20,
                                20, 17, 18, 20,
                                21, 18, 18, 19,
                                20, 20, 20, 20)
    fat2DTblFatContactMinCuts = (1, 2, 2, 2,
                                 2, 3, 3, 4,
                                 4, 4, 4, 6,
                                 4, 4, 4, 4)
}
```

Fat Metal Extension Contact Rule

The fat metal extension contact rule determines when the fat metal contact rule applies to normal wires that extend from a fat wire. In Figure 2-24 on page 2-30, the portion of the extension wire within the extension threshold, L, uses the fat metal contact rules.

To define a fat metal extension contact rule, specify the following attributes in a via `Layer` section of the technology file:

- `fatTblDimension`

  Specifies the size of the one-dimensional fat table.

- `fatTblThreshold`

  Specifies the thresholds used to determine which rules apply, based on the maximum width of the metal segments on the upper and lower metal layers. The width is the smaller of the x-length or y-length of the metal segment. You must specify *n* values, where *n* is the table dimension.

- `fatTblExtensionRange`

  Specifies the extension range thresholds. You must specify *n* values, where *n* is the table dimension.

- `fatTblExtensionContactNumber`

  Specifies the contact code numbers used to select vias from the library. You must specify *n* values, where *n* is the table dimension.

- `fatTblExtensionContactMinCuts`

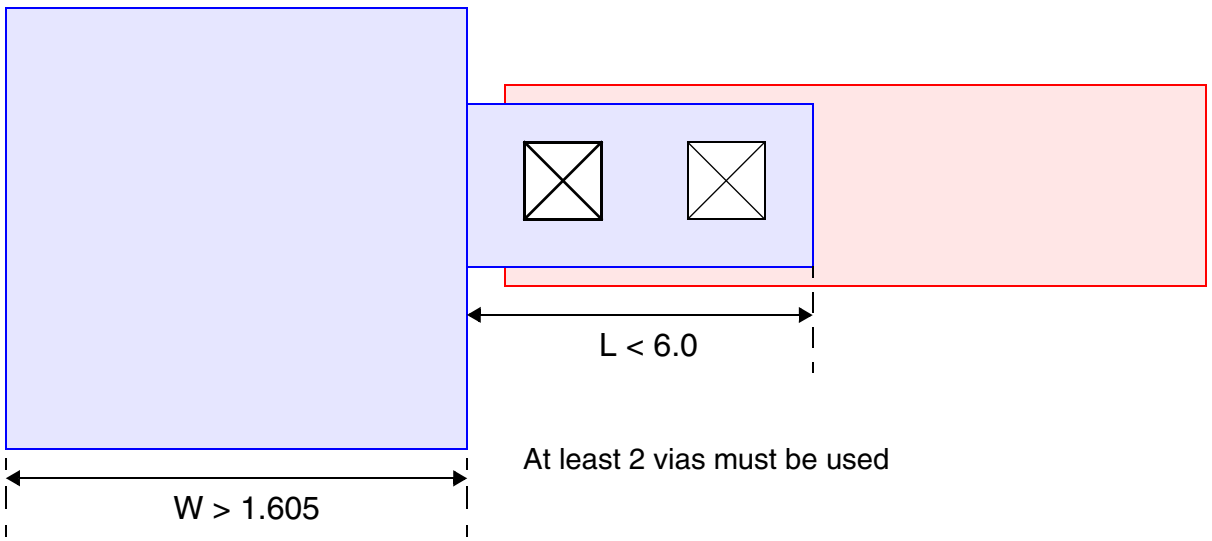
  Specifies the minimum number of vias for the wires within the extension range of fat wires. You must specify *n* values, where *n* is the table dimension.

For example,

```
layer   "VIA1" {
    fatTblDimension               = 3
    fatTblThreshold               = (0, 0.155, 1.605)
    fatTblExtensionRange          = (0, 0, 6)
    fatTblExtensionContactNumber  = (1, 21, 31)
    fatTblExtensionMinCuts        = (1, 1, 2)
}
```

In Figure 2-50, if the fat metal width is greater than 1.605 and the fat metal extension range is less than 6.0, contact code number 31 is used with at least 2 via cuts for a connection between the upper and lower metal layers.

Figure 2-50 Fat Metal Extension Contact Rule



Fat Poly Contact Rule

The fat poly contact rule specifies whether IC Compiler allows wires and vias that are connecting to pins to create fat shapes that would otherwise trigger fat via rule violations. The fat via rules are defined in the poly contact `Layer` section that connects the pin and metal layers.

To control the scope of the check for the fat poly contact rule, set the `dontMakePinFat` detail route option. You set this option with the `set_droute_options` command. The option setting is stored with the cell.

```
icc_shell> set_droute_options -name dontMakePinFat -value value
```

Setting the value to 0 causes only normal checking to be performed. Setting the value to 1 causes checking of all pin connections to avoid creating new fat shapes that could cause either `fatVia` or `minEnclosedArea` rule violations.

Via Enclosure Rules

The via enclosure rules are described in the following sections:

- Minimum Enclosure
- End-of-Line Via Enclosure Rules
- Two-Neighbor End-of-Line Via Enclosure Rule

Minimum Enclosure

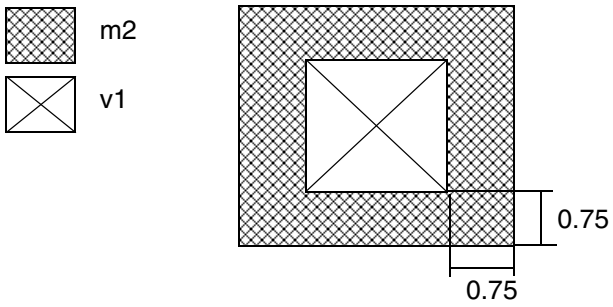
The minimum enclosure rule specifies the minimum distance at which layer1 must enclose layer2 when the two layers overlap.

This rule is defined in a `DesignRule` section of the technology file. To specify the minimum enclosure, use the `minEnclosure` attribute. For example, to specify that the metal 2 layer must enclose the via 1 layer by a minimum of 0.75 user units when the layers overlap, as shown in Figure 2-51, enter

```
DesignRule {
       layer1          = "M2"
       layer2          = "V1"
       minEnclosure    = 0.05
}
```

Figure 2-51 represents the minimum enclosure rule, which says that when the metal 2 and via 1 layers overlap, the metal 2 layer must enclose the via 1 layer at a distance of 0.75 user units.

Figure 2-51 minEnclosure Rule Example



End-of-Line Via Enclosure Rules

The end-of-line via enclosure rule applies when the via is in an end-of-line orientation and a rule that applies when the via is in a T-shape orientation. Both rules apply to single vias only, and not to double vias.

To specify whether the router checks or ignores these two rules, set the `ignoreMetalExtensionRule` detail route option. You set this option with the `set_droute_options` command. The option setting is stored with the cell.

```
icc_shell> set_droute_options -name ignoreMetalExtensionRule -value value
```

The following table lists the valid values for the `ignoreMetalExtensionRule` detail route option.

Value	Description
0	Checks the rules (the default)
1	Ignores the rules, even when they are set in the technology file

Metal Extension for End-of-Line Via Enclosure Rule

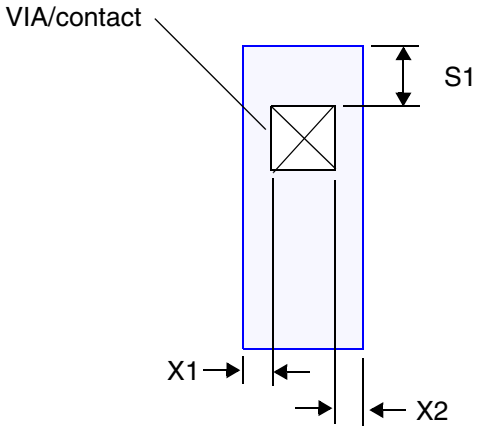
Use the `endOfLineEncTblSize`, `endOfLineEncSideThreshold`, and `endOfLineEncTbl` attributes to specify the end-of-line via enclosure rule. Define these attributes in the `DesignRule` section of the technology file.

For example,

```
DesignRule      {
   layer1                           = "VIA4"
   layer2                           = "M5"
   endOfLineEncTblSize              =  3
   endOfLineEncSideThreshold        = (0.005, 0.015, 0.025)
   endOfLineEncTbl                  = (0.081, 0.061, 0.000)
}
```

Figure 2-52 shows the parameters associated with the metal extension for end-of-line via enclosure rule.

Figure 2-52 Metal Extension for End-of-Line Via Enclosure Rule



When a single via/contact is in the end-of-line of upper metal or lower metal, the extension is greater than S1. The extension depends on X1 and X2; if X1 is different from X2, the smaller value is applied. Therefore,

- If 0.005 mm <= X1/X2 < 0.015 mm, S1 is greater than 0.08 mm for the via
- If 0.015 mm <= X1/X2 < 0.025 mm, S1 is greater than 0.06 mm for the via

T-Shape Metal Extension for End-of-Line Via Enclosure Rule

Use the `endOfLineTShapeEncTblSize`, `endOfLineTShapeCornerMinSpacing`, `endOfLineTShapeEncSideThreshold`, and `endOfLineTShapeEncTbl` attributes to specify the T-shape end-of-line via enclosure rule. Define these attributes in the `DesignRule` section of the technology file.

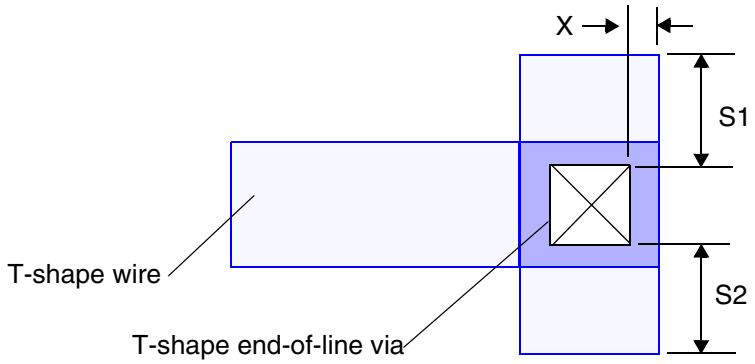
For example,

```
DesignRule  {
   layer1                           = "VIA4"
   layer2                           = "M5"
   endOfLineTShapeEncTblSize        = 4
```

```
    endOfLineTShapeCornerMinSpacing = 0.035
    endOfLineTShapeEncSideThreshold = (0.005, 0.010, 0.020, 0.035)
    endOfLineTShapeEncTbl           = (0.081, 0.061, 0.051, 0.041)
}
```

Figure 2-53 shows the parameters associated with the metal extension for end-of-line via enclosure rule.

Figure 2-53 T-Shape Metal Extension for End-of-Line Via Enclosure Rule



When a single via is in the end-of-line T-shape metal, the extension is greater than S1 and S2. The extension depends on X. Therefore,

- If X is less than or equal to 0.005 mm and less than 0.010 mm, S1 and S2 are greater than 0.08 mm for the via
- If X is less than or equal to 0.010 mm and less than 0.020 mm, S1 and S2 are greater than 0.06 mm for the via
- If X is less than or equal to 0.020 mm and less than 0.035 mm, S1 and S2 are greater than 0.05 mm for the via
- If X is less than or equal to 0.035 mm, S1 and S2 are greater than 0.04 mm for the via

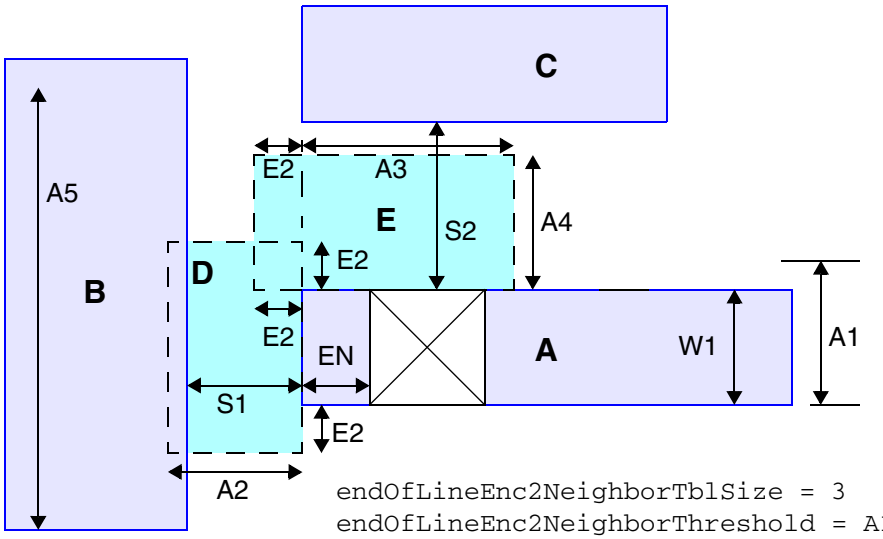
This rule applies only to a single via enclosed within a T-shaped wire. It does not apply to a double via. The rule can be disabled entirely by setting `ignoreMetalExtensionRule` to 1.

Two-Neighbor End-of-Line Via Enclosure Rule

The two-neighbor end-of-line via enclosure rule specifies the minimum spacing between an enclosed via in a narrow metal line and two metals near the line-end and along one side. In Figure 2-54, if the metal width W1 is less than or equal to the width threshold A1, then the distance between the line end and the first other metal S1 must be at least the minimum spacing value A2, or the distance between the side of the narrow metal line and the second other metal S2 must be at least the minimum side spacing value A4. Edges facing the end

of line edge are checked only if the edge length exceeds the wire minimum threshold A5. The line-end spacing value A4 is specified as an n-dimensional array, depending on the via enclosure value EN.

Figure 2-54 Two-Neighbor End-of-Line Via Enclosure Rule




```
endOfLineEnc2NeighborTblSize = 3
endOfLineEnc2NeighborThreshold = A1
endOfLineEnc2NeighborCornerKeepoutWidth = E2
endOfLineEnc2NeighborSideKeepoutLength = A3
endOfLineEnc2NeighborSideMinSpacing = A4
endOfLineEnc2NeighborTbl = (EN1, EN2, EN3)
endOfLineEnc2NeighborSpacingTbl = (A21, A22, A23)
endOfLineEnc2NeighborViaArrayExcludedTbl=(1, 1, 1)
endOfLineEnc2NeighborWireMinThreshold = A5
```

In the figure, either metal B must be outside of the dashed rectangle D or metal C must be outside of the dashed rectangle E. In this case, metal C is outside of dashed rectangle E, so the layout passes the rule. The dashed areas extend outward from the line-end corners by the corner keepout width E2. The side keepout area E has a length of A3 plus extension E2.

The rule is enforced only for a single via, and not for a via array, if the corresponding entry in the `endOfLineEnc2NeighborViaArrayExcludedTbl` table is set to 1. If the table entry is 0, the rule is enforced for both single vias and via arrays.

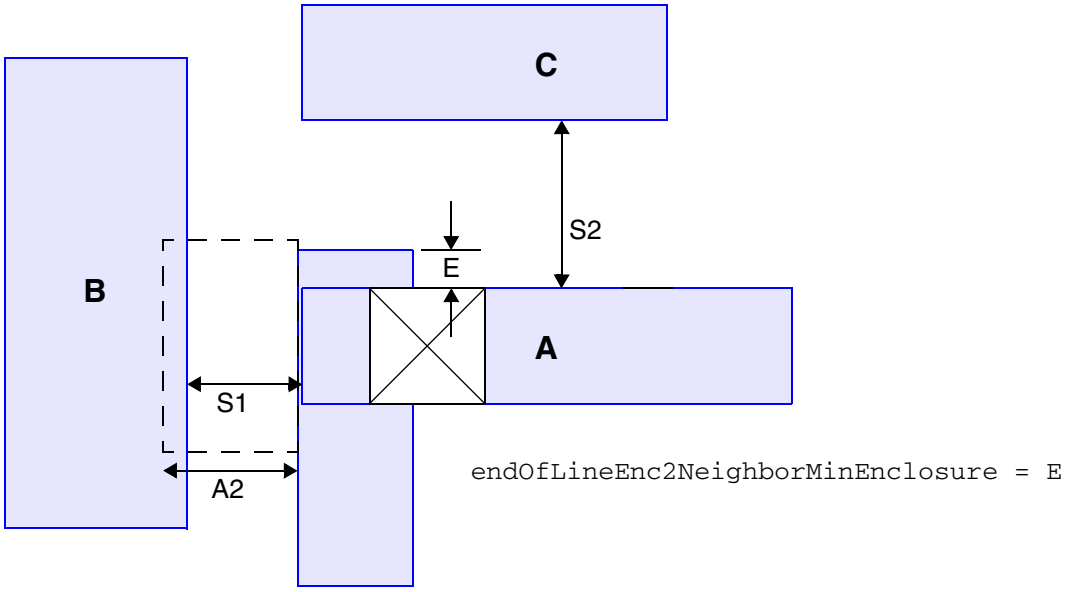
Here is an example of the syntax used for this rule:

```
DesignRule {
   layer1 = "M2"
   layer2 = "VIA1"
   endOfLineEnc2NeighborTblSize = 2
   endOfLineEnc2NeighborThreshold = 0.07
   endOfLineEnc2NeighborCornerKeepoutWidth = 0.03
   endOfLineEnc2NeighborSideKeepoutLength = 0.08
   endOfLineEnc2NeighborSideMinSpacing = 0.068
   endOfLineEnc2NeighborTbl = (0.038, 0.058)
   endOfLineEnc2NeighborSpacingTbl = (0.15, 0.13)
   endOfLineEnc2NeighborViaArrayExcludedTbl = (1, 1)
   endOfLineEnc2NeighborWireMinThreshold = .08
}
```

In this example, when a metal line having a width of less than 0.07 has neighboring metal at the end and at the side near the end, the spacing to the neighboring metal at the end must be at least the table-specified value of A2, which depends on the amount of via enclosure EN; or the spacing to the neighboring metal at the side must be at least 0.068. The keepout region extends 0.03 away from the line-end corners and 0.08 from the corner along the side.

You can also specify a minimum via enclosure on the sides perpendicular to the closer spacing, as shown in Figure 2-55.

Figure 2-55 Two-Neighbor End-of-Line Via Enclosure Rule



For example,

```
DesignRule {
   layer1 = "M2"
   layer2 = "VIA1"
   endOfLineEnc2NeighborTblSize = 2
   endOfLineEnc2NeighborThreshold = 0.07
   endOfLineEnc2NeighborCornerKeepoutWidth = 0.03
   endOfLineEnc2NeighborSideKeepoutLength = 0.08
   endOfLineEnc2NeighborSideMinSpacing = 0.068
   endOfLineEnc2NeighborMinEnclosure = 0.04
   endOfLineEnc2NeighborTbl = (0.038, 0.058)
   endOfLineEnc2NeighborSpacingTbl = (0.15, 0.13)
   endOfLineEnc2NeighborViaArrayExcludedTbl = (1, 1)
   endOfLineEnc2NeighborWireMinThreshold = .08
}
```

Note:

This rule is supported only by Zroute. For a similar rule supported by the classic router, see “Enhanced Dense End-of-Line Spacing Rule” on page 2-52.

Jog Wire Rules

The jog wire rules are described in the following sections:

- Small Jog Rule
- Jog Wire End-of-Line Via Rule
- Jog Wire Via Keepout Region Rule

Small Jog Rule

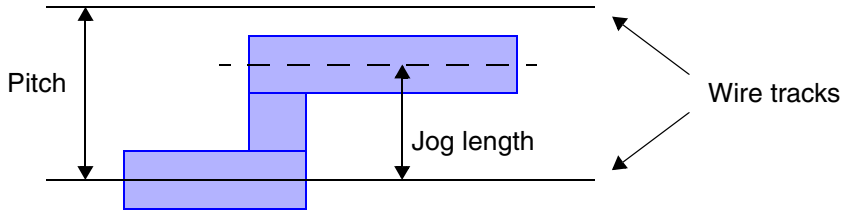
Use the `droute_smallJogMinLength` variable to specify the minimum length allowed for a small jog. The following table shows the valid values for this variable and their meanings.

Value	Description
0	The small jog rule is not checked.
1	The jog length needs to be greater than or equal to 1/4 pitch of the wire tracks for the routing layer.
2	The jog length needs to be greater than or equal to 1/2 pitch of the wire tracks for the routing layer.

When you specify a value of 1 or 2, the small jog rule is checked, and then fixed by the search-and-repair operation. A violation occurs when the length of the jog is shorter than 1 or 2 times the quarter pitch of the wire tracks.

Figure 2-56 shows the parameters associated with the small jog rule.

Figure 2-56 Small Jog Rule



For example, when you enter the following:

```
icc_shell> set droute_smallJogMinLength 2
```

a violation occurs when the jog is shorter than half the pitch of the wire track.

Jog Wire End-of-Line Via Rule

Use the `endOfLineViaJogWidth`, `endOfLineViaEncWidth` and `endOfLineViaJogLength` attributes to specify the jog wire end-of-line via rule. These attributes let you define additional location constraints for vias that connect to short jog wires. Define these attributes in a `DesignRule` section of the technology file.

Jog Wire Via Keepout Region Rule

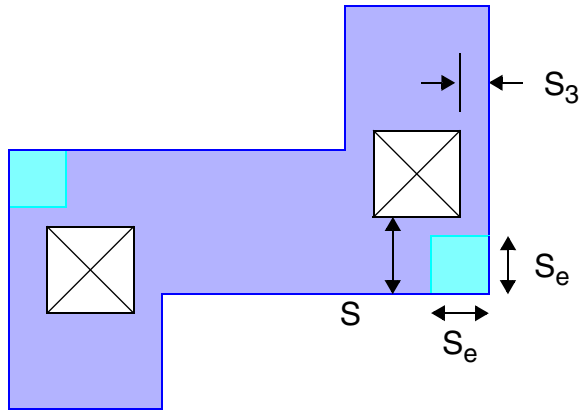
Use the `jogWireViaKeepoutTblSize`, `jogWireViaKeepoutEncThreshold`, and `jogWireViaKeepoutMinSize` attributes to specify the jog wire via keepout region rule. When you do not want a single via on a jog wire to be placed too close to the outside corner of the jog wire, use this rule to define a via keepout region at the jog wire corner. Define these attributes in the `DesignRule` section of the technology file, by specifying different values between the metal and via layer. For example,

```
DesignRule {
   layer1                          = "M1"
   layer2                          = "V1"
   minEnclosure                    = 0
   jogWireViaKeepoutTblSize        = 3
   jogWireViaKeepoutEncThreshold   = (0.005,0.015,0.025)
```

```
    jogWireViaKeepoutMinSize        = (0.08,0.06,0)
}
```

Figure 2-57 shows the parameters associated with the jog wire via keepout rule. In this example, when the via enclosure S_3 is less than 0.005, then the spacing to the corner S_e must be at least 0.08; the keepout region at the corner measures 0.08 by 0.08. When the via enclosure S_3 is greater than 0.005 but less than 0.015, then the spacing to the corner S_e must be at least 0.06.

Figure 2-57 Jog Wire Via Keepout Region Rule



This rule applies only to a single via enclosed within a jog wire. It does not apply to a double via in a given jog wire.

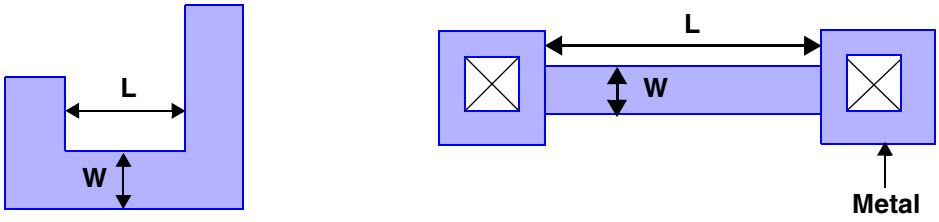
Dog Bone Rule

The dog bone rule applies during wire jogging and pin access when notches can appear next to narrow wires. A violation occurs when both of the following conditions exist:

- The metal width is less than the value specified with the `sameNetWidthThreshold` attribute (W < X).
- The notch spacing is less than the value specified with the `sameNetMinSpacing` attribute (L < Y).

Figure 2-58 shows the parameters associated with the dog bone rule.

Figure 2-58 Dog Bone Rule



Use the following attributes to specify the dog bone rule:

`sameNetWidthThreshold`

Specifies a number that represents the threshold value for the thin wire width.

`sameNetMinSpacing`

Specifies the minimum spacing between inside edges of the wire.

Define these attributes in a metal `Layer` section of the technology file.

For example,

```
Layer    "M5"   {
    sameNetWidthThreshold    = 0.5
    sameNetMinSpacing        = 1.0
}
```

Protrusion Length Rule

The protrusion length rule applies when a fat wire has both of the following:

- A width larger than a threshold value T specified with the `protrusionFatThresholdTbl` attribute
- A connected thin wire whose length is less than a value L specified with the `protrusionLengthLimitTbl` attribute

In such cases, the thin wire must have a width of at least the value W specified with the `protrusionMinWidthTbl` attribute.

Use the following attributes to specify the protrusion length rule:

`protrusionFatThresholdTbl`

Specifies a floating-point number that represents the threshold value for the fat wire.

`protrusionLengthLimitTbl`

Specifies a floating-point number that represents the length value for the connected thin wire.

`protrusionMinWidthTbl`

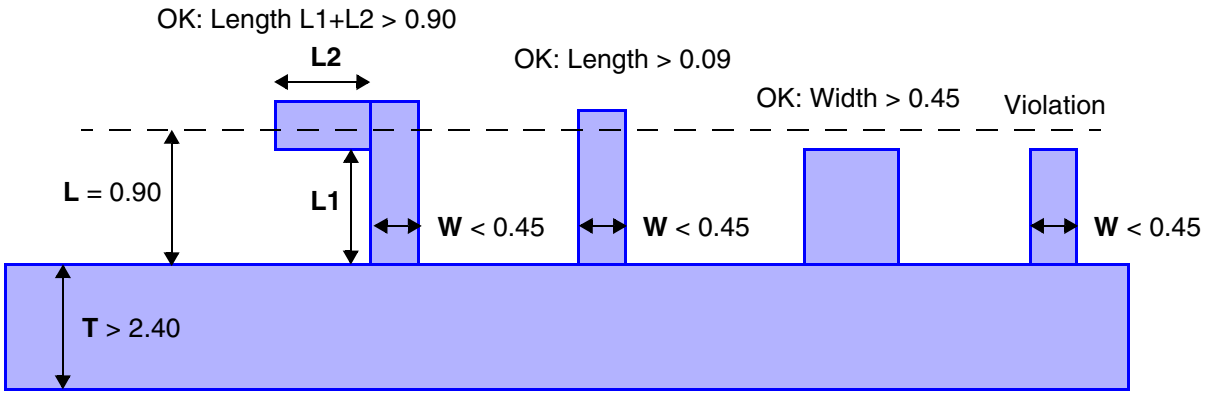
Specifies a floating-point number that represents the minimum width value for the connected thin wire.

Define these attributes in a metal `Layer` section of the technology file. For example,

```
Layer   "M2"  {
    protrusionTblDim       = 2
    protrusionFatThresholdTbl       = (1.20,2.40)
    protrusionLengthLimitTbl       = (0.60,0.90)
    protrusionMinWidthTbl = (0.30, 0.45)
}
```

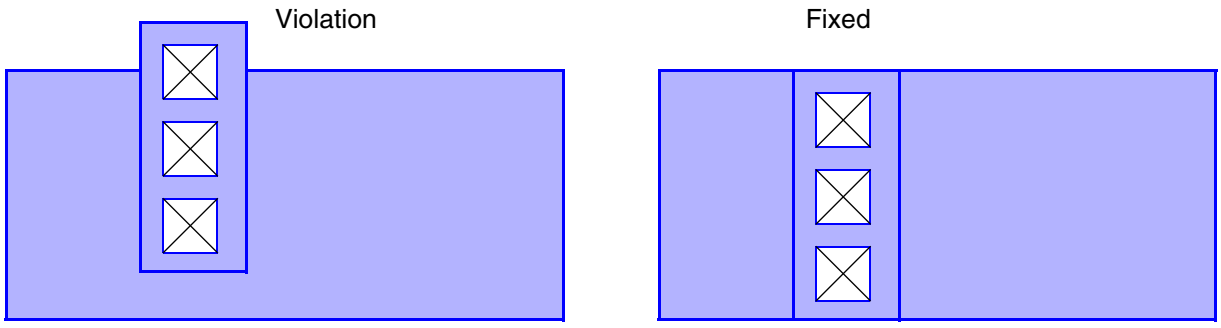
Figure 2-59 shows the parameters associated with the protrusion length rule.

Figure 2-59 Protrusion Length Rule



A protrusion length violation can occur where a wire is connected to a fat power or ground net by dropping a via or via array, as shown in Figure 2-60. A violation such as this can be fixed by adding metal stubs or rerouting the wire.

Figure 2-60 Protrusion Length Rule Violation and Correction



Via Maximum Stack Level Rule

The `maxStackLevel` attribute specifies the maximum number of vias in successive layers that can be stacked upon each other. If the number of successive stacked vias exceeds the specified limit, it is a rule violation.

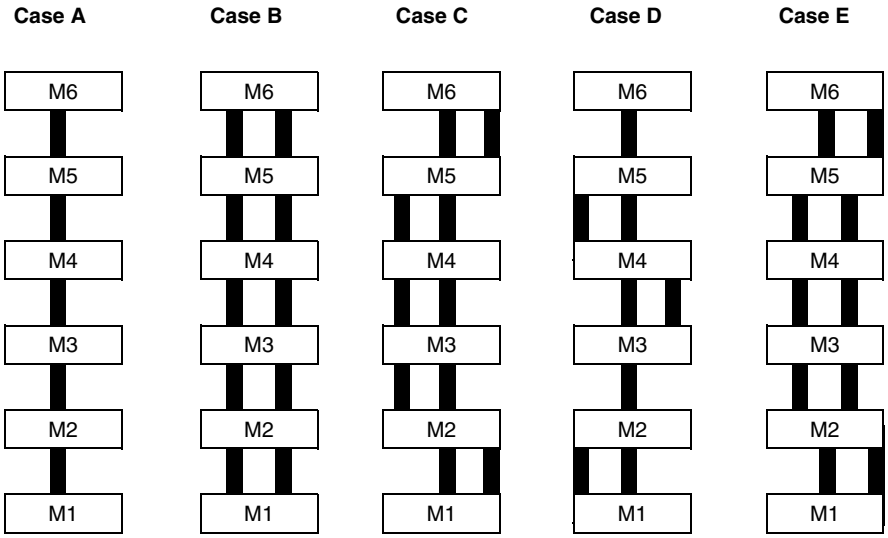
There are four operating modes that determine the scope of stacked via checking, which can depend on whether the vias are single or belong to an array. The `maxStackLevelMode` attribute in the `Technology` section of the technology file specifies the checking mode. You can set this attribute to a number from 0 to 3, defined as follows:

- 0 (the default): The rule applies only to a set of stacked vias when some or all of the vias are single. The rule is ignored when all of the stacked vias belong to arrays.
- 1: The rule applies to all stacked vias, whether single or belonging to an array.
- 2: The rule applies only to a set of stacked vias when all of them are single vias. The rule is ignored if any of the stacked vias belong to an array.
- 3: This rule applies to all stacked vias except in the case where the stacked vias all belong to via arrays and the arrays are aligned in the stack.

For example,

```
Technology   {
   maxStackLevelMode = 1
   }

Layer "Via12" {
   maxStackLevel = 4
   }
```

The following figure and table show the scope of the maximum stack level check for each checking mode when you specify a `maxStackLevel` value of 4. The figure shows a cross-section view of the metal layers and black vias between them. The presence of two side-by-side vias indicates a via array. The table indicates whether a violation is detected for each checking mode, 0 through 3, and each case, A through E, based on the number of stacked vias, the alignment of the vias, and whether the vias are single or arrayed.



checkViaArrayMaxStackLevel mode	Case A	Case B	Case C	Case D	Case E
0	Violation			Violation	
1	Violation	Violation	Violation	Violation	
2	Violation				
3	Violation			Violation	

Fat Metal Via Keepout Rules

The fat metal via keepout rules are specified in the `DesignRule` section of the technology file by using the `fatWireViaKeepoutTblSize`, `fatWireViaKeepoutWidthThreshold`, `fatWireViaKeepoutMinSize`, `fatWireViaKeepoutEnclosure`,

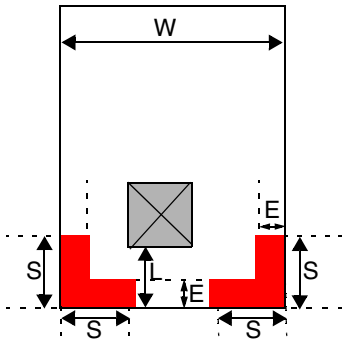
`fatWireViaKeepoutParallelLengthThreshold`, and `fatWireViaKeepoutMaxSpacingThreshold` attributes. You can use these attributes to define the following rules:

- The fat metal via keepout area rule
- The via enclosure rule
- The poly contact enclosure rule

Fat Metal Via Keepout Area Rule

When you do not want the via at the end-of-line upper-fat wire or lower-fat wire to be placed too close to the corner of the fat wire, use this rule to define a via keepout region at the end-of-line corners (shown in red in Figure 2-61). A single via must be placed outside of the keepout area. In the case of a double via, at least one of the vias must be placed outside of the keepout area.

Figure 2-61 Fat Metal Via Keepout Area Rule



When the width, W, of the fat wire is greater than or equal to the `fatWireViaKeepoutWidthThreshold` value, the distance, L, between the via and the corner edges of the fat wire must be in greater than or equal to the `fatWireViaKeepoutEnclosure` value, E. The length of the keepout area in each direction from the corner, S, is specified by the `fatWireViaKeepoutMinSize` attribute.

For example,

```
DesignRule     {
   layer1                           = "V1"
   layer2                           = "M2"
   fatWireViaKeepoutTblSize         = 2
   fatWireViaKeepoutWidthThreshold = (0.5,1.0)
   fatWireViaKeepoutMinSize         = (0.18,0.36)
   fatWireViaKeepoutEnclosure       = (0.05,0.10)
}
```

```
DesignRule     {
   layer1                           = "V1"
   layer2                           = "M1"
   fatWireViaKeepoutTblSize         = 1
   fatWireViaKeepoutWidthThreshold  = (0.5)
   fatWireViaKeepoutMinSize         = (0.18)
   fatWireViaKeepoutEnclosure       = (0.05)
}
```

Fat Via Enclosure Rule

The fat via enclosure rule defines the minimum width, G, of the metal enclosure of a via under the following conditions:

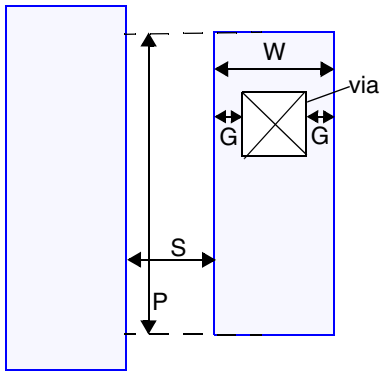
- The width, W, of the enclosing metal segment is between the widths specified in the `fatWireViaKeepoutWidthThreshold` attribute.
- The space, S, between the enclosing metal segment and the neighboring metal segment is less than the value specified in the `fatWireViaKeepoutMaxSpacingThreshold` attribute.
- The parallel length, P, between the enclosing metal segment and the neighboring metal segment is greater than the value specified in the `fatWireViwKeepoutParallelLengthThreshold` attribute and the parallel run covers the poly contact edge partially or completely.
- The enclosing metal does not contain a double-via array.

Use the `fatWireViaKeepoutEnclosure` attribute to specify the minimum width of the metal enclosure.

The `pinRectangleMerge` detail route option controls how pin and obstacle rectangles are merged when checking this rule. When set to 0 (the default), neither pin nor obstacle rectangles are merged. When set to 1, pin rectangles, but not obstacle rectangles, are merged. When set to 2, both pin and obstacle rectangles are merged.

Figure 2-62 shows the measurements involved in the fat via enclosure rule.

Figure 2-62 Via Enclosure Rule



For example, the following `DesignRule` section defines a minimum enclosure width of 0.015 microns when the enclosing metal 2 segment is

- Between 0.11 and 0.21 microns wide
- Closer than 0.08 microns to its neighboring metal 2 segment
- Parallel to its neighboring metal 2 segment for at least 0.27 microns

```
DesignRule      {
   layer1                           = "M2"
   layer2                           = "VIA"
   fatWireViaKeepoutParallelLengthThreshold = 0.27
   fatWireViaKeepoutMaxSpacingThreshold = 0.08
   fatWireViaKeepoutTblSize         = 2
   fatWireViaKeepoutWidthThreshold = (0.11,0.21)
   fatWireViaKeepoutMinSize         = (2,2)
   fatWireViaKeepoutEnclosure       = (0.015,0)
}
```

```
icc_shell> set_droute_options -name pinRectangleMerge -value 2
```

Fat Poly Contact Enclosure Rule

The fat poly contact enclosure rule is the same as the fat via enclosure rule, but it defines the enclosure requirements between a poly contact and a metal segment, rather than between a via and a metal segment. This rule does not apply if the enclosing metal contains a double poly contact array.

For example, the following `DesignRule` section and option settings define a minimum enclosure width of 0.015um when the enclosing metal 1 segment is between 0.11 µm and 0.21 µm wide, is closer than 0.08 µm to its neighboring metal segment, and is parallel to it for at least 0.27 µm.

```
DesignRule      {
   layer1                           = "M1"
   layer2                           = "CO"
   fatWireViaKeepoutParallelLengthThreshold = 0.27
   fatWireViaKeepoutMaxSpacingThreshold = 0.08
   fatWireViaKeepoutTblSize         = 2
   fatWireViaKeepoutWidthThreshold = (0.11,0.21)
   fatWireViaKeepoutMinSize         = (2,2)
   fatWireViaKeepoutEnclosure       = (0.015,0)
}
```

```
icc_shell> set_droute_options -name pinRectangleMerge -value 2
```

Via-on-Grid Rule

You can force vias to be placed on the routing grid. For Zroute, use the following syntax in the technology file:

```
Layer V1 {
          onGrid = 1
}
```

This example forces V1 vias to be placed on the grid.

For the classic router, you can use the `noOffGridRouting` detail route option to specify that all vias above metal layer M2 must be on grid. Enter

```
icc_shell> set_droute_options -name noOffGridRouting -value  4
```

Keep the following points in mind:

- Only vias must be on grid. Wires can be routed off grid to allow them to touch off-grid pins.
- V12 can be off grid to allow wires to touch off-grid pins.

Use the `noOffGridRouting` detail route option and the `Vn`*`n`*`NoOffGridRouting` detail route options to specify a layer-by-layer control for checking that vias are on grid, where *n* specifies the layer. You must set `noOffGridRouting` to 4 and set `Vn`*`n`*`NoOffGridRouting` to 1 for each layer that requires the vias to be on grid.

The syntax is

```
set_droute_options -name Vn NoOffGridRouting -value value
```

Value	Description
0	Ignores the via-on-grid check for the specified layers
1	Checks that vias are on grid for the specified layers (the default)

For example, when V2 through V5 are required to be on grid, use the following commands:

```
icc_shell> set_droute_options -name noOffGridRouting -value 4
icc_shell> set_droute_options -name V1NoOffGridRouting -value 0
icc_shell> set_droute_options -name V2NoOffGridRouting -value 1
icc_shell> set_droute_options -name V3NoOffGridRouting -value 1
icc_shell> set_droute_options -name V4NoOffGridRouting -value 1
icc_shell> set_droute_options -name V5NoOffGridRouting -value 1
icc_shell> set_droute_options -name V6NoOffGridRouting -value 0
...
icc_shell> set_droute_options -name V12NoOffGridRouting -value 0
```

Keep the following points in mind:

- The grid must be defined as a preferred-direction-to-preferred-direction grid. The rule does not consider nonpreferred direction grids.
- The via-on-grid rule checks the upper-layer preferred tracks and lower-layer preferred tracks only.

Via Array (Via Farm) Rule

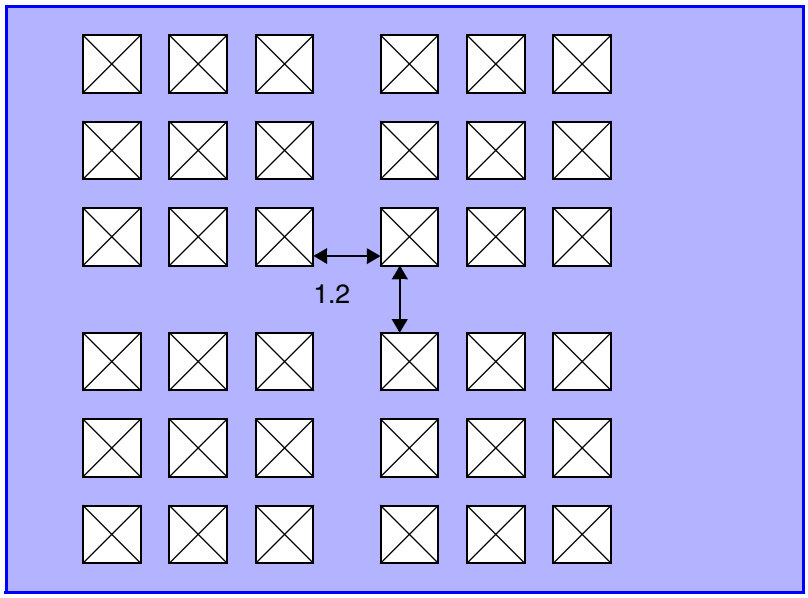
The via farm rule specifies the maximum number of rows in a via array and the minimum spacing between two via arrays. This rule is used only by the power and ground router.

This rule is defined in the `ContactCode` section of the technology file. Each via array requires a `ContactCode` definition. To specify the via array size, use the `maxNumRows` attribute. To specify the spacing between two via arrays, use the `viaFarmSpacing` attribute. For example,

```
ContactCode "VIA12_fat" {
   maxNumRows     = 3
   viaFarmSpacing = 1.2
}
```

In this example, the maximum via array size is 3 by 3 and the minimum spacing between via arrays is 1.2, as shown in Figure 2-63.

Figure 2-63 Via Farm Rule



By default, the maximum number of rows and via farm spacing constraints apply to both the horizontal and vertical directions. To apply these rules only in the direction of the longer wire intersections, set the `viaFarmLongDirection` attribute to 1. In this case, the number of rows and spacing constraints in the shorter direction are determined by the intersection boundaries and minimum cut spacing. For example,

```
ContactCode "VIA12_fat" {
   maxNumRows           = 3
   viaFarmSpacing       = 1.2
   viaFarmLongDirection = 1
}
```

Parallel Length-Based Floating Wire Antenna Rule

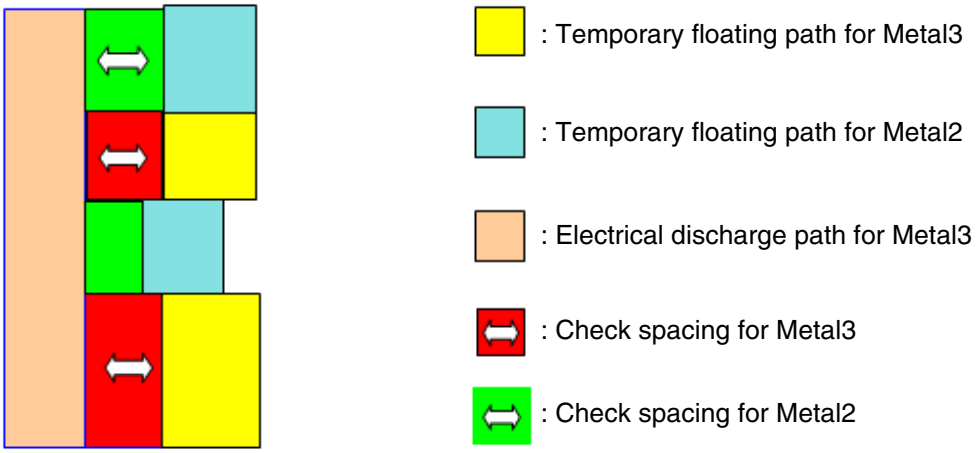
IC Compiler checks floating wire antennas by checking the maximum allowable value for metal areas and the minimum spacing to the adjacent wire. The parallel length-based floating wire antenna rule also considers the parallel length between the floating wire and the adjacent wire. Use the following options to specify the parallel length-based floating wire antenna rule:

```
M1FloatingWirePLength1-5
```

```
M2FloatingWirePLength1-5
...

M12FloatingWirePLength1-5

M1FloatingWirePLMinSpc1-5

M2FloatingWirePLMinSpc1-5
...

M12FloatingWirePLMinSpc1-5
```

The floating wires will be merged according to the values you specify.

For example, in the following figure, the Metal2 and Metal3 floating wires will be merged.



Metal Density Rules

This section describes the metal density rule and the metal density gradient rule.

Metal Density Rule

Use the following attributes to specify the metal density rule:

`layer`

Defines the metal layer for which the rule is specified.

windowSize

Defines the size of the window for the density check. By default, the window step size is half of the defined windowSize.

minDensity

Defines the minimum percentage of metal allowed in the window.

maxDensity

Defines the maximum percentage of metal allowed in the window.

These attributes are defined in the DensityRule section of the technology file. Each metal layer requires a DensityRule definition. For example,

```
DensityRule {
   layer =  "M1"
   windowSize =  200
   minDensity =  20
   maxDensity =  80
}
```

The metal fill command, insert_metal_filler, honors the metal density rule. That is, the total percentage of the named metal layer in the window size will be within the specified minimum and maximum density limits after you run insert_metal_filler.

Metal Density Gradient Rule

Use the densityGradient attribute to specify that the fill density between adjacent windows should not be more than the density gradient specified. Define this attribute in the DensityRule section of the technology file.

The density gradient information is read in during the insert_metal_filler operation. The fill density of each window is compared with the corresponding density of its neighbors. If the percentage difference between them exceeds the density gradient of that layer, the fill is trimmed to satisfy the density gradient rule.

Via Density Rule

Use the following attributes to specify the via density rule:

layer

Defines the via layer for which the rule is specified.

windowSize

Defines the size of the window for the density check. By default, the window step size is half of the defined windowSize.

`minDensity`

Defines the minimum percentage of metal allowed in the window.

`maxDensity`

Defines the maximum percentage of metal allowed in the window.

These attributes are defined in the `DensityRule` section of the technology file. Each via layer requires a `DensityRule` definition. For example,

```
DensityRule {
   layer =  "VIA1"
   windowSize =  200
   minDensity =  2
   maxDensity =  5
}
```

Index

E

F

G

I

L

M

P

R

S

T

U

V